N° d'ordre :

THÈSE

présentée à

L'UNIVERSITÉ DE PARIS-SUD
CENTRE D'ORSAY

pour obtenir

LE TITRE DE DOCTEUR DE 3ème CYCLE

par

Jean-Luc ALBIN

Spécialité : **Mathématiques**

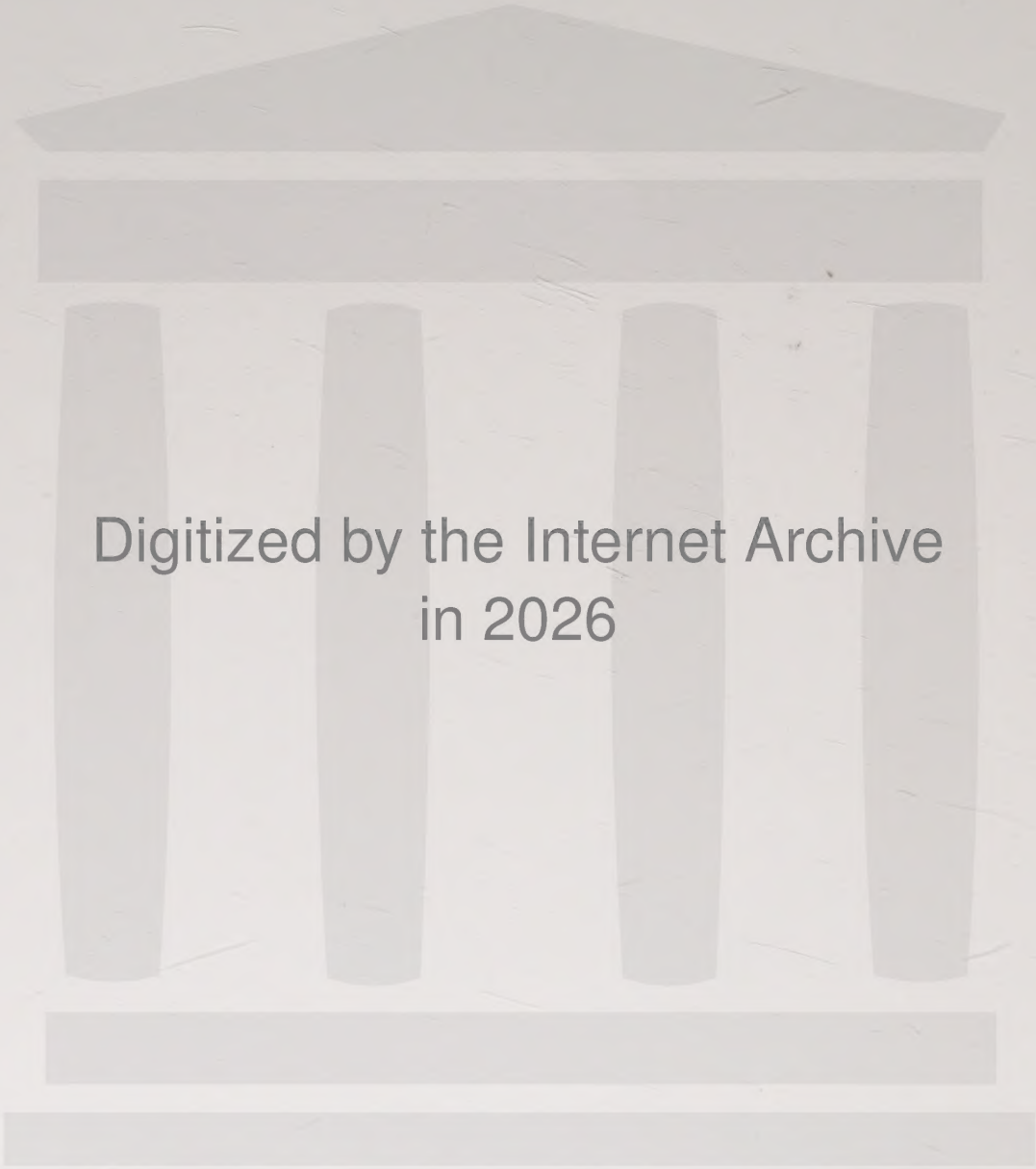
Mention : **Informatique**

Sujet de la thèse :

ÉVALUATION DE LA FIABILITÉ DU LOGICIEL À L'AIDE DE L'ANALYSE STATISTIQUE DE SES MESURES DE COMPLEXITÉ

Soutenue le 17 Novembre 1982 devant la Commision composée de :

MM.	**E. GELENBE**	Président
	D. POTIER	Examinateurs
	E. GIRARD	
	J.C. RAULT	Invités
	G. RUGET	


Digitized by the Internet Archive
in 2026

https://archive.org/details/IA41547318_0004

THÈSE

présentée à

L'UNIVERSITÉ DE PARIS-SUD
CENTRE D'ORSAY

pour obtenir

LE TITRE DE DOCTEUR DE 3ème CYCLE

par

Jean-Luc ALBIN

Spécialité : Mathématiques
Mention : Informatique

Sujet de la thèse :

ÉVALUATION DE LA FIABILITÉ DU LOGICIEL À L'AIDE DE L'ANALYSE STATISTIQUE DE SES MESURES DE COMPLEXITÉ

Soutenue le 17 Novembre 1982 devant la Commision composée de :

MM.	E.	GELENBE	Président
	D.	POTIER	Examinateurs
	E.	GIRARD	
	J.C.	RAULT	Invités
	G.	RUGET	

UNIV.
LEIDEN
BIBL.

Je tiens à remercier,

Monsieur E.Gelenbe, Professeur à Paris XI, de l'honneur qu'il me fait en présidant le jury de ma thèse.

Monsieur D.Potier, Ingénieur de Recherches, de m'avoir intéressé à ce sujet, accueilli dans son équipe à l'INRIA, aidé à mener à bien cette étude et permis de m'initier à différents domaines statistiques et informatiques.

Monsieur E.Girard, Ingénieur à la CIMSA, d'avoir lu et jugé ce travail.

Monsieur J.Cl.Rault, Ingénieur à l'ADI, d'accepter de participer au jury.

Monsieur G.Ruget, Professeur à Paris XI, du plaisir qu'il me fait en s'intéressant à ce travail et en étant membre du jury.

Je remercie spécialement R.Férréol, Ingénieur à la CIMSA, de la totale disponibilité dont il fit preuve. Son enseignement et les fructueux échanges d'idées que j'ai eus avec lui m'ont été très utiles lors de la réalisation finale de ce rapport.

Je remercie aussi toutes les personnes qui m'ont aidé à divers titres durant ces deux années; en particulier A.Bilodeau, Ingénieur à ECA-Automation, M.Badel, Y.Lechevallier et A.Schroeder, Ingénieurs à l'INRIA.

Cette thèse marque la fin de mes études universitaires. En signe de reconnaissance, je la dédie à mes parents.

PAPIER RECUPÉRÉ E

Ce travail a été financé par l' Agence de l'Informatique (ADI). Il s'inscrit dans le cadre du projet pilote SURF (SUReté de Fonctionnement des systèmes), et correspond à un contrat passé par l'ADI auprès de

- la Compagnie d'Informatique Militaire Spatiale et Aéronautique (CIMSA),
- ECA-Automation,
- l'Institut National de Recherches en Informatique et Automatique (INRIA).

Résumé

Cette thèse présente une étude quantitative d'évaluation statique de la fiabilité d'un logiciel industriel.

Le premier chapitre rappelle, à travers un exposé bibliographique, les différentes méthodes d'évaluation de la fiabilité d'un logiciel et détaille les fondements de l'approche suivie dans cette étude.

Ensuite les résultats d'une expérience de saisie et d'analyse des données caractéristiques du développement et de la complexité d'un logiciel important sont présentés. Plus précisément, les points étudiés dans les quatre chapitres suivants sont, respectivement, les procédures de collecte des données, l'analyse des données d'erreurs, l'analyse des mesures de complexité et la recherche de liaisons entre les caractéristiques de fiabilité du logiciel étudié et ses propriétés de complexité.

En conclusion, l'utilisation pratique de cette étude et sa portée sont évoquées, quelques suggestions sont avancées.

Mots clés

fiabilité logicielle, mesures de complexité, analyse statistique, génie logiciel.

Abstract

This thesis presents an experiment in collecting and analysing industrial software data.

The first chapter presents the different methods of software reliability evaluation and points out the bases of the approach followed in this study.

The next four chapters examine, respectively, the data collection procedures, the analysis of software error data, the analysis of complexity data and the investigation of significant relationships between complexity measures and software reliability.

The practical aspect of this study and its scope are evocated and some suggestions are proposed in the conclusion.

Key words

software reliability, complexity measures, statistical analysis, software engineering.

CHAPITRE 1

1 APPLICATION DE LA THEORIE DE LA FIABILITE DU MATERIEL

1.1 Modèle de Jelinski et Moranda

1.2 Modèle de Schick et Wolverton

1.3 Modèle de Musa

2 MESURES STATIQUES DE PROGRAMMES

2.1 Complexité textuelle

2.2 Complexité structurelle

2.2.1 nombre cyclomatique

2.2.2 nombre de chemins et atteignabilité

2.3 Autres études expérimentales

2.3.1 autres définitions de mesures

2.3.2 résultats sommaires d'une étude bibliographique

2.4 Conclusions

CHAPITRE 1

Le matériel a bénéficié des progrès récents réalisés dans le domaine de la microélectronique, en taille et en coût des composants. Cela a entrainé une mise en lumière tardive de l'importance du logiciel sur l'ensemble d'un système informatique. Des chiffres révèlent en effet le coût croissant du logiciel par rapport au matériel.

> Le budget informatique de l'U.S. AIR FORCE s'est réparti en 1960 entre 75% au matériel et 25% au logiciel; en 1973 80% était consacré au logiciel; en 1985 il est prévu que le logiciel représentera environ 90% du coût total des systèmes informatiques.

Une analyse de ces coûts montre l'écart entre le coût de l'écriture du logiciel et celui de sa mise au point.

> En 1972 le département Avionics de l'U.S AIR FORCE a consacré 75$ par instruction pour l'écriture et jusqu'à 4000$ par instruction erronée pour la correction de ses programmes <Pi76>. D'autres études confirment que le coût de correction de programmes par rapport à celui de test peut être dans le rapport 45 <DACS79>.

De plus la maintenance occupant une partie non négligeable d'un personnel que l'on soustrait à des taches de création nuit au développement.

Tous ces faits ont conduit les concepteurs de logiciel à se préoccuper du problème de sa fiabilité. La bibliographie de Génie Logiciel montre que les nombreuses recherches d'évaluation de la fiabilité du logiciel relèvent de deux approches différentes:

- une approche qualitative
- une approche quantitative

L'approche qualitative se définit globalement par des efforts axés sur la définition de meilleures méthodologies, l'élaboration de techniques de développement portant sur des sujets divers comme les langages et les techniques de programmation, l'écriture et la validation des spécifications, la preuve formelle de la correction des programmes, les procédures de test, la gestion des projets informatiques,...

L'ouvrage de Rault <Ra80> précise davantage ces notions, il permet de montrer que ces efforts représentent le résultat d'une démarche, en deux temps, prise habituellement pour obtenir un fonctionnement fiable du logiciel. En premier lieu, éviter les erreurs à priori; ce que certains auteurs désignent par l'intolérance aux défauts. On recherche alors l'élimination de tous les défauts avant toute utilisation du programme. En dépit de nombreux efforts importants, cette démarche n'a jamais permis d'atteindre une bonne fiabilité. Cette constatation a conduit à utiliser d'autres voies et en particulier celle de la tolérance aux défauts. La présence d'erreurs dans un logiciel est considérée alors comme inéluctable.

Cette approche plus réaliste impose des conditions à l'architecture d'un système pour que celui-ci puisse tolérer des défauts, citons par exemple le confinement des erreurs (limiter la propagation des effets d'une erreur), les mécanismes de reconfiguration, les mécanismes de reprise, etc. Les techniques de tolérance peuvent être classées en deux catégories:

- les techniques de la redondance: elles utilisent des mécanismes tels que l'altération éventuelle des divers composants du système en cas d'apparition d'erreurs est soit corrigée soit éliminée. On entend par redondance toute partie de code, de données, d'exécution non strictement nécessaire à l'obtention des résultats. Plusieurs mécanismes sont présentés dans <MeRa79, Ra80>.
- les techniques de la protection: elles utilisent des mécanismes par lesquels on tente d'éviter l'altération éventuelle des divers composants du système. Il s'agit alors plutôt de mesures préventives.

Le détail des techniques de protection (les différents environnements possibles, leurs avantages et leurs inconvénients, la mise en oeuvre des techniques, les systèmes où sont implantées ces techniques,...), le détail des techniques de redondance (les différents mécanismes, leurs performances et les résultats expérimentaux) sont fournis de facon complète dans <MeRa79, Ra80>.

La seconde approche, l'évaluation quantitative de la fiabilité du logiciel, correspond davantage aux besoins des concepteurs et des utilisateurs de posséder des procédures formelles. Les premiers ont besoin de connaître quels sont les facteurs prépondérants de la fiabilité. La mise en oeuvre de procédures correspondantes dans le processus de conception et un choix rationnel parmi les méthodologies de conception possibles constituent une voie d'approche vers la fiabilité. Les utilisateurs souhaitent connaître de plus en plus la confiance que l'on peut attribuer à un produit logiciel qu'ils viennent d'acquérir en ce qui concerne la fiabilité.

Le concept d'évaluation de la fiabilité concerne l'expression quantitative de la confiance que l'on peut attribuer à un programme. En fait il n'existe pas de mesures absolues qui, selon certains, auraient aucune signification à cause des définitions particulières de la fiabilité que l'on peut retenir. Une liste de ces définitions figure dans <MeRa79, Ra80>. Il est plutôt préférable de parler de guides de techniques quantitatives d'évaluation, chaque technique correspondant à une définition de la fiabilité. Si l'on considère les principes sous-jacents, les techniques de mesures peuvent être classées en deux catégories:

- celles résultant d'extensions, au cas du logiciel, des techniques utilisées pour l'évaluation de la fiabilité du matériel,
- celles spécifiques du logiciel, prenant en compte la complexité interne des programmes.

Le présent chapitre suit cette classification.

1 APPLICATION DE LA THEORIE DE LA FIABILITE DU MATERIEL

L'évaluation de la fiabilité du matériel ayant fait l'objet d'expérimentations convaincantes, il est tentant de vouloir définir la fiabilité du logiciel et du matériel de façon semblable (même concept, même vocabulaire, notion de temps moyen entre défauts, de taux de défaillance).

Une définition précise de la fiabilité d'un programme issue de la théorie de la fiabilité du matériel peut-être trouvée dans <Pi76>.

> La fiabilité d'un programme est la probabilité que, sur la machine pour laquelle il a été concu et dans les conditions prévues dans ses spécifications, ce programme fonctionne sans erreur pendant un temps donné.

Cette approche de la fiabilité repose sur le taux d'apparition d'erreurs. La fiabilité s'estime alors, si Z(x) est le taux de défaillance estimé ou mesuré, par

$$F(t) = \exp\int_0^t -(Z(x)*dx)$$

Au cours des différentes études d'estimation de la fiabilité du logiciel au moyen des techniques propres au matériel, une quinzaine de modèles ont été développés mais trois principalement ont été bien validés. Ils se distinguent par leurs jeux d'hypothèses <Pi76, MeRa79>.

1.1 Modèle de Jelinski et Moranda

hypothèses:

- les défauts sont détectés aléatoirement,
- le taux de détection est constant entre les occurences de deux erreurs consécutives,
- les défauts sont détectés indépendamment les uns des autres,
- les défauts sont corrigés immédiatement après leur détection,
- la correction d'un défaut n'en entraine aucun autre,
- le taux de détection est proportionnel au nombre de défauts résiduels.

calculs: Nous donnons en détail les principaux calculs probabilistes. Ils seront repris, mais sans explications, pour les deux autres modèles. Notons:

- N le nombre total de défauts,
- n le nombre de détection de défauts,
- Z(i) le taux de défaillance entre les défauts i et i-1,
- x(i) la durée s'écoulant entre la détection des défauts i et i-1.

Nous obtenons:

$$Z(i) = K*(N-i+1)$$

De cette formule, les valeurs de K et N sont inconnues, il faut les estimer. Nous allons chercher la fonction du maximum de vraisemblance du modèle. Pour cela, on peut prouver que la fonction de densité f(x(i)) vaut:

$$f(x(i)) = Z(i) * \exp -(Z(i)*x(i))$$

Par hypothèse, les défauts sont détectés indépendamment, la fonction du maximum de vraisemblance s'écrit:

$$L(x(1),\ldots,x(n)) = \prod_{1}^{n} f(x(i))$$

Les estimateurs de K et N sont solutions des équations

$$\partial L/\partial K = 0 \qquad \text{et} \qquad \partial L/\partial N = 0$$

La moyenne des temps de bon fonctionnement (MTBF) après la détection de p défauts est, en notant Ke et Ne les estimateurs de K et N

$$MTBF(p) = \int_{0}^{\infty} \exp -Ke*(Ne-(p+1)+1)*x(p+1) \quad d(x(p+1)) = 1/(K*(N-p))$$

De même, avec une MTBF donnée, égale à m, on doit détecter:

$$p = N-(K*m)**(-1)-n \text{ défauts}$$

1.2 Modèle de Schick et Wolverton

Les hypothèses 1 à 5 du modèle précédent sont conservées. Par contre les équipes de programmation se perfectionnant avec le temps l'hypothèse 6 devient:

> le taux de détection entre les occurences de deux défauts consécutifs i-1 et i est proportionnel au nombre d'erreurs résiduelles et au temps écoulé depuis la détection du défaut i-1.

Soit

- x le temps écoulé depuis la détection du i-1 défaut,
- x(i) le temps écoulé entre la détection des défauts i et i-1.

On déduit

$$Z(x(i)) = K*(N-i+1)*x(i)$$

La densité de probabilité est:

$$f(x(i)) = Z(x(i))*\exp\{-K*(N-i+1)*(x(i)**2)/2\}$$
$$E(x(i)) = \{3.14 / (2K*(N-n+1))\}**(1/2) = MTBF(i-1)$$

La fonction du maximum de vraisemblance est:

$$L(x(1),\ldots,x(n) = \prod_{1}^{n} f(x(i))$$

Les estimateurs du maximum de vraisemblance de K et de N sont solutions du système

$$\partial L/\partial K = 0 \quad \text{et} \quad \partial L/\partial N = 0$$

1.3 Modèle de Musa

hypothèses:

- les erreurs sont indépendantes entre elles et sont distribuées selon une loi uniforme,
- le temps d'exécution entre deux erreurs est grand par rapport au temps moyen d'exécution d'une instruction,
- le test est représentatif d'une utilisation du programme,
- les données d'entrée sont tirées au hasard,
- toutes les conséquences d'une erreur sont observables,
- toute erreur détectée est corrigée immédiatement.

Si

- f représente la fréquence d'exécution moyenne des instructions,
- n le nombre de défauts corrigés au temps t.

On déduit:

$$Z(t) = K*f*(N-n)$$
$$dn/dt = Z(t)$$

car d'après les dernières hypothèses le taux de correction d'erreurs dn/dt doit être égal au taux de défaillance. On déduit:

$$n = N*(1- \exp(-fKt))$$

La fiabilité est alors caractérisée par sa MTBF, T, égale à 1/Z(t) soit:

$$T = (\exp fKt)/(f*K*N).$$

Cependant l'approche quantitative de la fiabilité du logiciel à partir des techniques propres au matériel est souvent accueillie avec scepticisme par les spécialistes du logiciel car des différences importantes subsistent (pas de notion d'usure dans le cas du logiciel, les défauts sont de nature différente).

Le principe de base des techniques spécifiques au logiciel que nous avons retenues est de relier une mesure de la fiabilité à des mesures de la complexité des programmes. L'hypothèse faite est l'existence de paramètres caractéristiques de programmes. Bien que l'influence de la

complexité des programmes sur le coût total des produits logiciels soit un fait reconnu depuis longtemps, des définitions formelles de la complexité assorties de mesures datent seulement d'environ six, huit ans. Deux types de complexité correspondant à une analyse statique des programmes seront présentés.

2 MESURES STATIQUES DE PROGRAMMES

Toutes les mesures statiques sont issues directement ou indirectement de la forme codée des programmes. Par opposition aux mesures dynamiques elles définissent la complexité d'un programme sans l'exécuter. L'ensemble des mesures statiques peut être partitionné en deux catégories suivant le type de mesures considéré:

- mesures de complexité textuelle,
- mesures de complexité structurelle.

2.1 Complexité textuelle

La Science du Logiciel s'appuie sur l'hypothèse que les caractéristiques physiques des programmes, tels son vocabulaire, sa taille, son nombre de bits necéssaires, le temps de programmation, l'effort, le niveau du programme, ... obéissent à certaines lois, comme il en existe en Thermodynamique. Toutes les caractéristiques de programmes sont définies au moyen des quatre mesures textuelles suivantes <Ha77>:

- le nombre d'opérateurs distincts n1,
- le nombre d'occurences des opérateurs N1,
- le nombre d'opérandes distinctes n2,
- le nombre d'occurences des opérandes N2.

Les paramètres physiques utilisés dans la Science du logiciel, définis plus loin, sont:

- le vocabulaire n,
- la taille N,
- le volume Vol,
- le niveau du programme L,
- la difficulté D,
- le niveau du langage LAMBDA,
- l'effort E,
- le temps T,
- le nombre d'erreurs B.

Halstead a estimé la taille d'un programme connaissant son vocabulaire par:

$$Ne = (n1*\log(n1) + n2*\log(n2))/\log(2)$$

Cette formule suppose le programme bien structuré. Pour l'appliquer, il faut éliminer les impuretés des programmes, soient:

- les opérations complémentaires,
- les opérandes ambigus,
- les opérandes synonymes,
- les sous expressions communes,
- les assignations non nécessaires,
- les expressions non factorisées.

Le volume d'un programme est défini par:

$$V = (N1+N2)*\log(n1+n2)/\log(2)$$

Cette formule se justifie intuitivement. Il faut log(n1+n2)/log(2) bits pour coder les opérateurs et les opérandes distincts du texte donc pour exprimer les (N1+N2) mots du programme choisis parmi les opérateurs ou opérandes, il faut V bits.

Un programme donné ayant un volume V dans un langage donné aura un volume différent dans un autre langage. On pourrait imaginer un langage le plus évolué possible pour lequel il n'y aurait que deux opérateurs (la fonction à réaliser et l'assignation) et N2 opérandes d'entrée/sortie. Si le langage était suffisamment complet tout programme serait aussi simple que cela. On appelle volume idéal, noté V*, le volume du programme écrit en langage de plus haut niveau; et niveau du programme le rapport de ce volume idéal au volume réel du programme écrit. Halstead a proposé un estimateur de cette mesure comprise entre 0 et 1

$$L = (2/n1) * (n2/N2) = 1/D$$

Lorsqu'un algorithme est traduit d'un langage dans un autre son volume idéal reste constant. De facon similaire, lorsqu'un algorithme écrit en un langage déterminé est modifié, on constate que le volume idéal et le niveau de programme sont inversement proportionnels: leur produit définit une constante du langage appelée niveau du langage. L'expression de cette variable est

$$LAMBDA = L*(V*) = L*L*V$$

L'effort mental exigé par la fabrication d'un programme a été mesuré par Halstead en "unités élémentaires de raisonnement".

$$E = V/L = V*D$$

Halstead fait l'hypothèse que le temps de programmation est proportionnel à l' effort. Le coefficient de proportionnalité représente la "vitesse" du programmeur qu'il estime à dix-huit au moyen de plusieurs programmes de différentes tailles. Le psychologue Stroud <St66> a établi qu'un esprit humain est capable d'assimiler de cinq à vingt discriminations mentales par seconde selon ses capacités. La valeur de la vitesse du programmeur a été ensuite confirmée sur d'autres données <Ste66, Hu73>. Le temps de programmation s'estime par:

$$T = E/18$$

Halstead a cherché à établir une relation numérique entre l'effort et le nombre d'erreurs contenues dans le programme. On peut penser que ce dernier est proportionnel au volume, ainsi avec K constante de proportionnalité, on peut écrire:

```
B = K*V
B = K * E**(2/3) * LAMBDA**(1/3)
B = K'* E**(2/3)
```

Sur les données d'Akiyama on obtient:

```
B = E**(2/3) / 3200
```

Une autre technique d'analyse statique de programmes, spécifique au logiciel, définie à l'aide du graphe de contrôle des programmes et développée au milieu des années 70, est souvent utilisée dans les études expérimentales de mesures de complexité.

2.2 Complexité structurelle

Etant donné un programme, il est possible d'associer un graphe orienté G avec une entrée et une sortie uniques. Chaque sommet correspond à un ensemble d'instructions purement séquentielles (ie: sans instruction de contrôle, ni étiquette). Un arc joint un sommet X à un sommet Y si, et seulement si, l'exécution des instructions du sommet X peut immédiatement précéder celle des instructions du sommet Y. G est appelé graphe de contrôle. Afin de rendre G fortement connexe, on ajoute un arc de retour de la sortie vers l'entrée de G.

2.2.1 nombre cyclomatique

Le nombre cyclomatique représente la mesure fondamentale issue de la complexité structurelle, il est défini de la façon suivante <McC76>: si le graphe possède e arcs, n sommets et p composantes connexes, alors le nombre cyclomatique du graphe de contrôle vaut:

$$nu = e-n+p$$

en pratique p=1, car il est évident que le graphe est connexe.

On démontre, dans le cas d'un graphe fortement connexe G, que le nombre cyclomatique est égal au nombre de tests binaires du programme plus 1.

$$nu = T+1$$

Cette relation permet de calculer simplement le nombre cyclomatique.

De plus, dans le cas d'un ensemble de programmes le nombre cyclomatique du programme global est la somme des nombres cyclomatiques de chaque composante (la complexité résultante d'un programme est la somme des complexités de chaque procédure).

Enfin un programme structuré a un nombre cyclomatique pouvant être ramené à 1. Un programme est structuré s'il n'y a pas de sauts à l'extérieur ou à l'intérieur d'une boucle, à l'extérieur ou à l'intérieur d'un test <McC76>.

Une fréquente critique est portée à la mesure de Mac Cabe:

> deux boucles imbriquées et deux boucles qui se succèdent donneront un graphe de contrôle de même complexité, ce qui va contre l'intuition et l'expérience.

Deux autres mesures ont été définies dans la bibliographie, elles présentent l'intérêt de ne pas être directement issues du nombre cyclomatique mais du graphe de contrôle, contrairement à beaucoup d'autres. Ces mesures corrigent la critique formulée à l'encontre du nombre cyclomatique puisqu'elles identifient deux boucles (tests) en séquence comme structure plus complexe que deux boucles (tests) emboitées. Ces deux mesures sont le nombre de chemins et l'atteignabilité du graphe de contrôle <ScHo79>.

2.2.2 nombre de chemins et atteignabilité

La mesure du nombre de chemins est prise avec la restriction d'un seul parcours par boucle ce qui semble légitime puisque répéter une boucle n'apporte pas de complexité supplémentaire. L'atteignabilité est une mesure moins intuitive. Sa définition consiste à calculer à chaque sommet le nombre de chemins incidents (ne pas confondre avec le nombre d'arcs) puis à sommer sur tous les sommets. Ces deux définitions ont été rarement utilisées pour étudier la complexité de différents logiciels <FeFo79, ScHo79>.

2.3 Autres études expérimentales

2.3.1 autres définitions de mesures

Myers estime insuffisante la définition du nombre cyclomatique et propose de définir un intervalle de complexité, <My77>:

$$I = \langle \text{nb instr. contrôle} + 1 ; \text{nb cond.simples} + 1 \rangle$$

Mais Hansen critique la mesure de Myers car les deux termes ne sont pas indépendants. Il propose de mesurer la complexité du graphe et la complexité de l'expression du programme et suggère le couple, <Han78>:

$$I = \langle \text{nb instr. conditionnelles} ; \text{nb opérateurs} \rangle$$

mais comment classer deux intervalles I1=<a;b> et I2=<c;d> tels que a<c et b>d?. Aucune étude expérimentale ne permet de déterminer si ces deux mesures sont pertinentes.

Une quantité importante d'autres mesures a été définie. La plupart soit difficiles à mettre en oeuvre soit sans signification précise autre que leur formulation mathématique n'ont pas fait l'objet d'une étude expérimentale approfondie. Citons néammoins la mesure de Woodwart <Wo79> de compilation des noeuds du graphe réduit, un noeud consistant en l'existence de deux arcs non orientés (X,Y) et (Z,T) ; X<Y Z<T tels que:

X<Z<Y<T ou Z<X<T<Y

La portée d'une variable, (dernière référence - première référence)/(nombre de références-1) et la densité de décisions, rapport du nombre cyclomatique par le nombre de noeuds sont deux mesures faciles à mettre en euvre mais aucune étude expérimentale permet d'évaluer leur pertinence <FeFo79>, de même les mesures de Culik et de Chapin restent à expérimenter <Cu79, Ch79>.

2.3.2 résultats sommaires d'une étude bibliographique

Pour la communauté logicielle, il est fréquent de considérer les notions de qualité d'un programme et de productivité d'une équipe en termes de lignes de codes (LOC) écrites. Ainsi en est-il des temps d'écriture, du nombre de programmeurs nécéssaires à la réalisation d'un programme, du coût d'un produit. Halstead suggère une approche différente, basée sur les opérateurs et les opérandes du texte. Une autre technique s'est développée autour de la notion de graphe de contrôle de programmes. Dans le but d'établir les mérites respectifs des trois mesures

- effort,
- LOC,
- nombre cyclomatique.

comme meilleur prédicteur des facteurs les plus déterminants dans l'élaboration d'un projet logiciel, une importante bibliographie prouve qu'indépendamment du critère retenu, production de logiciel fiable, nombre d'erreurs, ou effort de maintenance requis (durée de détection d'erreurs, de correction) la variable d'Halstead effort caractérise mieux un logiciel que le nombre cyclomatique, la mesure LOC étant la variable la moins pertinente, ceci indépendamment du langage utilisé, de la taille et des structures des programmes, de la compétence des programmeurs <FiLo79, CuSh79, Su81, FeFo79, FuHa76, ScHo79>. Néammoins ces études mettent en évidence de fortes corrélations parmi ces trois mesures. Une présentation exhaustive des travaux qui prouvent ces conclusions n'est pas l'objet de ce chapitre. Une bibliographie complète peut être trouvée dans <Ra80>.

La conclusion de l'étude comparative des mesures de complexité structurelle constitue la deuxième observation bibliographique intéressante <ScHo79>. Schneidewind et Hoffmann montrent que le nombre cyclomatique, le nombre de chemins et l'atteignabilité caractérisent aussi bien leurs données d'erreurs issues d'un système de données de la marine Américaine. Toutefois parcequ'il est plus facile à mettre en oeuvre, toujours fini et qu'il évoque une notion de programmation significative en mesurant le nombre de

constructions indépendantes, ils suggèrent d'adopter de préférence le nombre cyclomatique.

2.4 Conclusions

L'exposé bibliographique indique que les mesures de la Science du Logiciel et les mesures issues de la complexité structurelle sont des paramètres de complexité d'un produit logiciel, par ailleurs ces mesures peuvent être aisément intégrées au processus de conception de logiciels. Les besoins des utilisateurs et des concepteurs d'évaluer la fiabilité du logiciel seront couverts si des relations liant les mesures de complexité de programmes à la fiabilité sont établies. Pour atteindre cet objectif des méthodes élaborées d'analyse statistique ont été utilisées. Les principaux résultats de toutes les méthodes statistiques sont donnés en annexe.

Le chapitre suivant présente le logiciel analysé.

CHAPITRE 2

1 PRESENTATION DES DONNEES D'ERREURS

1.1 Définition et structure du compilateur LTR

1.1.1 la souche

1.1.2 le générateur

1.2 Méthode de production du compilateur LTR

1.2.1 la souche

1.2.2 le générateur

1.3 Collecte des erreurs et codage des informations

1.3.1 collecte des erreurs

1.3.2 codage de l'information

1.4 Typologies employées

1.4.1 typologie d'erreurs

1.4.2 typologie d'origine et de détection

2 MESURES DE COMPLEXITE

2.1 Structure de l'outil de mesures

2.2 Définition des mesures de complexité

2.2.1 mesures textuelles

2.2.2 mesures structurelles

2.3 Méthodes de prélèvement des mesures de complexité structurelle

2.4 Conclusions

CHAPITRE 2

Les outils statistiques employés dans les différentes études expérimentales de mesures de complexité ou de fiabilité du logiciel sont souvent: moyennes, corrélations, régression, ou s'appliquent généralement à un jeu restreint de données et la portée des résultats s'en trouve limitée <FeFo79, MoBr77, ScHo79, Sm80, Su81>. La réalisation du logiciel analysé demanda plusieurs années de travail et permit, dans le cadre d'une politique de suivi de programmes, de rassembler un grand nombre d'informations et de constituer un important fichier de données d'erreurs.

Ce chapitre présentera le logiciel analysé et les données d'erreurs enregistrées dans un premier paragraphe, les mesures de complexité dans un second.

1 PRESENTATION DES DONNEES D'ERREURS

Le programme analysé est le compilateur LTR IRIS-50 écrit en LTR et développé à la CIMSA.

1.1 Définition et structure du compilateur LTR

Le rôle d'un compilateur est de traduire un langage de haut niveau, ici le langage LTR (proche du PASCAL), en un langage assembleur. Le développement en parallèle de plusieurs compilateurs:

- compilateur LTR sur MITRA,
- compilateur LTR sur IRIS 50,
- compilateur LTR sur IRIS 80,
- compilateur LTR sur MPR.

a conduit les réalisateurs à concevoir les compilateurs LTR sous la forme d'une partie commune à tous les compilateurs, responsable surtout de la traduction du code LTR en code intermédiaire (nous appellerons cette partie commune la souche), et d'une partie, spécifique à la machine sur laquelle il est destiné, chargée de la génération du code machine (nous appellerons cette partie le générateur). Les compilateurs LTR, en particulier le compilateur LTR IRIS 50, sont donc structurés en deux parties distinctes:

1.1.1 la souche

La souche analyse le programme écrit en LTR, elle est indépendante de la machine cible mais dépendante du langage LTR. Elle produit des tables et des textes intermédiaires qui traduisent sous forme codée l'ensemble des informations contenues dans le programme. Dans le cas du compilateur IRIS-50, la souche est composée de sept unités logiques (appelées unités fonctionnelles) qu'il est possible de caractériser de la façon suivante:

- précompilateur noté Z,
- analyseur lexicale A,
- analyseur syntaxique B,C,
- analyseur sémantique D,
- prégénérateur E,
- optimiseur F,

1.1.2 le générateur

Le générateur fait la synthèse des informations fournies par la souche. Il dépend de la machine cible puisqu'il réalise la traduction des tables et des textes intermédiaires dans l'assembleur de la machine concernée. Dans le cas du compilateur LTR IRIS-50 il est composé de quatre unités fonctionnelles qu'il est possible de caractériser ainsi:

- générateur 1 G,
- générateur 2 H,
- générateur de données K,
- générateur d'assembleur L.

La figure 2.1 décrit la structure du compilateur LTR IRIS-50 (les ARTICLES DE DONNEES sont semblables aux instructions Fortran BLOCK-DATA).

1.2 Méthodes de production du compilateur LTR

Outre leurs fonctionnalités distinctes, les méthodes de production de la souche et du générateur furent différentes.

1.2.1 la souche

La souche du compilateur LTR IRIS-50 est une extension de la "souche" du premier compilateur LTR, destiné d'ailleurs à l'IRIS 50. Ce dernier compilateur était composé d'un seul bloc: si les différences de fonctionnalité entre la souche et le générateur ont été bien mises en évidence, les interfaces entre les deux parties n'ont pas été suffisamment spécifiées pourque l'on puisse admettre que la structure de ce compilateur LTR était souche-générateur. La souche de l'actuel compilateur LTR IRIS 50 n'a jamais été reprise, et donc à fortiori réanalysée. Les nouvelles fonctions inhérentes à la réalisation d'une nouvelle souche ont été intégrées à la "souche" du premier compilateur LTR par adjonction de codage. Il est donc aventureux de parler d'une véritable méthode d'analyse de la souche. L'intégration de cet élément fonctionnel dans différents compilateurs a permis de nombreux tests en exploitation.

1.2.2 le générateur

Le générateur du compilateur LTR ne fut pas développé à partir d'un générateur existant. Un travail de spécifications globales de l'architecture du générateur fut d'abord entrepris, ensuite les spécifications plus détaillées et la réalisation furent confiées à chaque membre de l'équipe de développement dont les qualifications au départ étaient d'ailleurs assez bonnes et indiscutablement meilleures que celles de

l'équipe de la souche. La spécificité du générateur, lié à un seul type de machine cible, a entrainé que les tests en exploitation ont été moins poussés que dans le cas de la souche.

1.3 Collecte des erreurs et codage des informations

1.3.1 collecte des erreurs

Une politique de suivi de programmes peut permettre le recueil d'un nombre important d'informations relatives aux données d'erreurs. Au moyen d'une part de fiches suiveuses, qui contiennent les informations relatives à la préanalyse de l'erreur et les éléments permettant l'identification de la correction, au moyen d'autre part de fiches de corrections, qui possèdent les éléments nécessaires à l'identification de l'erreur par rapport à la fiche suiveuse ainsi que toutes les indications permettant d'effectuer la correction, 1224 erreurs ont été notées. La collecte des informations relatives aux données d'erreurs s'est faite à l'aide d'un programme de saisie.

1.3.2 codage de l'information

Pour chaque erreur, ces informations sont codées sur 72 caractères. Le schéma d'enregistrement du fichier de données est détaillé à la figure 2.2. Il correspond aux quatre champs suivants:

- l'unité source erronée (nom du programme, unité fonctionnelle, nom de la procédure) 0—29 caractères.
- l'anomalie (numéro du rapport d'anomalie, date de création de la fiche suiveuse, date de création de la fiche de correction) 30—48 caractères.
- l'erreur (niveaux de détection et d'origine de l'erreur, sous types de l'erreur) 49—56 caractères
- la correction (numéro de la fiche de correction, type de correction) 57—71 caractères.

Parmi les 1224 erreurs du compilateur, 49 soit mal relevées, soit mal codées n'ont pu être identifiées.

1.4 Typologies employées

1.4.1 typologie d'erreurs

La typologie d'erreurs classiques en fiabilité du logiciel et correspondant aux erreurs de Calcul, Logique, Interface, Manipulation de données,... n'a pas permis de couvrir l'ensemble des erreurs du compilateur LTR. Des codes complémentaires d'erreurs spécifiques au logiciel étudié ont donc été ajoutés. Ils correspondent à des erreurs:

- d'Optimisation (une optimisation au niveau du code généré nécessitant une transformation du compilateur),
- d'Outils utilisés (outils de mise au point dont l'incorporation dans le compilateur crée des anomalies),

- dans un autre langage (le compilateur possède certaines procédures écrites en Assembleur).

La codification complète des anomalies est donnée à la figure 2.3: les définitions classiques d'erreurs commencent par les codes A, B, ..., H; les définitions spécifiques au compilateur par les codes O, X, Y, Z.

1.4.2 typologie d'origine et de détection

Les niveaux d'origine d'erreurs localisent le fait générateur de l'anomalie observée et les niveaux de détection précisent la phase du projet au cours de laquelle l'anomalie est découverte. La définition de ces différents niveaux est donnée à la figure 2.4, ils couvrent la durée de vie d'un produit logiciel.

2 MESURES DE COMPLEXITE

Le logiciel d'analyse de qualité et de mesures (LAQME) a été réalisé par ECA-Automation dans le cadre du contrat SURF <BiPo80>. La présentation de l'outil de mesures, la définition des mesures de complexité retenues, les techniques de prélèvement des mesures de complexité structurelle constituent les trois parties de ce deuxième paragraphe.

2.1 Structure de l'outil de mesures

L'outil de mesures de complexité employé reçoit en entrée un fichier source LTR compilable et fournit en sortie le profil syntaxique et la valeur des différentes mesures de complexité utilisées. Eventuellement (en option), à la fin de chaque sous procédure ou de chaque procédure, un programme de recensement est appelé. A la fin du fichier un recensement général est fait pour tout le programme. Ce programme occupe 29K mots pour le code exécutable et 40K mots pour les données. Il peut analyser des fichiers de plus de 15000 cartes au rythme d'environ 100 cartes/seconde.

L'outil de mesures est composé de deux unités distinctes.

- L'unité d'analyse lit le programme et produit les paramètres nécessaires aux calculs sur les données, elle élabore aussi le graphe de contrôle. Plus précisément un premier bloc d'analyse syntaxique lit les déclarations, insère les variables et les constantes dans les tables d'identificateurs et les tables de constantes, compte les occurences des variables, des constantes, des séparateurs et des mots clés pour évaluer les quatre paramètres de base de la Science du Logiciel.
  Les piles syntaxiques nécessaires à la construction du graphe de contrôle sont gérées dans un deuxième bloc d'analyse syntaxique. Le graphe de contrôle, mis sous forme intermédiaire, est figuré dans un tableau de structures de trois entiers. Chaque structure définit un sommet du graphe, le premier entier identifie le sommet en donnant son numéro d'instructions, les deux autres entiers donnent l'adresse des sommets successeurs. L'instruction

CASE...OF est traitée différemment.
- La deuxième unité est l'unité de synthèse, elle transforme le graphe de contrôle afin que différents algorithmes de mesures soient appliqués.

Une forte hypothèse a été formulée sur le graphe de contrôle: il correspond à un programme structuré, c'est à dire le nombre cyclomatique de ce programme peut être ramené à 1 (cf définition CHAPITRE 1). La programmation structurée donne lieu à des programmes plus lisibles et plus faciles à maintenir. En outre, les analyseurs de programmes sont beaucoup plus simples à concevoir si l'on suppose qu'ils s'appliqueront à des programmes structurés. Pour un langage évolué comme le LTR, la programmation structurée est si répandue qu'elle tend à devenir une norme de programmation.

Sur les données du compilateur LTR, une forte proportion de procédures étaient structurées et l'hypothèse était donc vérifiée. Dans le cas contraire, les valeurs des deux mesures de complexité, définies au CHAPITRE 1 paragraphe 2.2.2:

- nombre de chemins,
- atteignabilité,

diffèrent. L'hypothèse formulée sur le graphe de contrôle affecte donc la notion de nombre de chemins lorsque la procédure n'est pas structurée. Or une analyse portant sur d'autres études expérimentales montre que la valeur du nombre de chemins d'une procédure non structurée n'est jamais calculée: soit elle est minorée pour des raisons de coût de calcul, soit l'on modifie lors du calcul du nombre de chemins la contribution des instructions qui rendent la procédure non structurée. Nous avons choisi la seconde possibilité de l'alternative. L'argument qui justifie ce choix est simplement le suivant: l'analyseur de complexité fournit toujours les valeurs de deux mesures de complexité, que la procédure soit structurée ou non. Lorsque la procédure est structurée, la majorité des cas, ces deux mesures de complexité représentent sémantiquement le nombre de chemins et l'atteignabilité. Lorsque la procédure n'est pas structurée, il faut comprendre ces deux mesures comme des indices de complexité (pour plus de détails, la figure 2.5 donne la méthode explicite de calcul du nombre de chemins).

2.2 Définitions des mesures de complexité

2.2.1 mesures textuelles

Les mesures textuelles ont été prélevées sur le code source. Le langage LTR, proche du PASCAL, possède six classes d'opérateurs

- opérateurs arithmétiques, logiques, booléens,
- opérateurs d'initialisation, d'assignation,
- opérateurs de parenthésage,
- opérateurs de GOTO étiquettes,
- mots clés (WHILE-DO, UNTIL-DO, FOR, BEGIN-END, IF-THEN-ELSE, IF-THEN),
- noms de variables (procédures,fonctions).

et deux classes d'opérandes

- les variables simples, tableaux, structures, ensembles,
- les constantes arithmétiques, logiques.

2.2.2 mesures structurelles

Les mesures structurelles sont effectuées sur le graphe de contrôle. Le graphe de contrôle se définit en faisant correspondre chaque sommet du graphe à un bloc d'instructions purement séquentielles et les arcs aux branchements du programme. Certains auteurs utilisent une définition légèrement différente: chaque sommet du graphe correspond à une instruction simple du programme et chaque arc à un branchement implicite ou non. Nous appelerons un tel graphe un graphe étendu car il comporte en général un nombre de sommets supérieur. Quatre mesures de complexité structurelle ont été employées:

- nu le nombre cyclomatique,
- V le nombre cocyclomatique,
- R l'atteignabilité,
- P le nombre de chemins.

mais deux d'entre elles dépendent de la définition du graphe de contrôle. Elles sont donc aussi mesurées sur le graphe de contrôle étendu:

- V' le nombre cocyclomatique étendu,
- R' l'atteignabilité étendue.

Ainsi, pour chaque procédure nous disposons de six mesures de complexité structurelle.

2.3 Méthodes de prélèvement des mesures de complexité structurelle

Dans le cas général, les méthodes combinatoires du calcul du nombre de chemins et de l'atteignabilité appliquables à partir de la forme intermédiaire du graphe de contrôle fournie par l'unité d'analyse demandent un temps calcul qui croît exponentiellement avec la taille de la procédure analysée. Pour pallier cette difficulté, Mac Cabe <McC76> propose une méthode de calcul du nombre de chemins associant une expression algébrique à chaque construction structurée et compte un parcours pour une instruction élémentaire (une assignation par exemple). L'atteignabilité peut être calculée en même temps puisqu'on connait le nombre de chemins incidents sur chaque sommet. Enfin, le calcul du nombre cyclomatique peut s'établir à partir des contributions cyclomatiques correspondant à chaque construction syntaxique.

Plus précisément la figure 2.5 présente, dans le cas du langage LTR, une liste étendue des constructions utilisées en programmation structurée (sont ajoutés le sommet initial et l'instruction if--then), les graphes correspondants, les contributions cyclomatiques et les expressions associées au calcul du nombre de chemins. Toutes ces relations sont utilisées par l'analyseur de programmes. Le graphe de contrôle généré par l'analyseur de programmes ne sert donc pas au calcul des mesures de

complexité structurelle, il est davantage conçu comme outil d'aide à la documentation des programmes.

2.4 Conclusions

L'outil de mesures de complexité peut effectuer une analyse de la procédure seule ou du programme complet. L'outil fournit en sortie le profil syntaxique du bloc de code analysé et le vecteur de ses mesures de complexité. Nous disposons pour chaque procédure et pour chaque unité fonctionnelle des quatre mesures de base de la complexité textuelle et de six mesures de complexité structurelle.

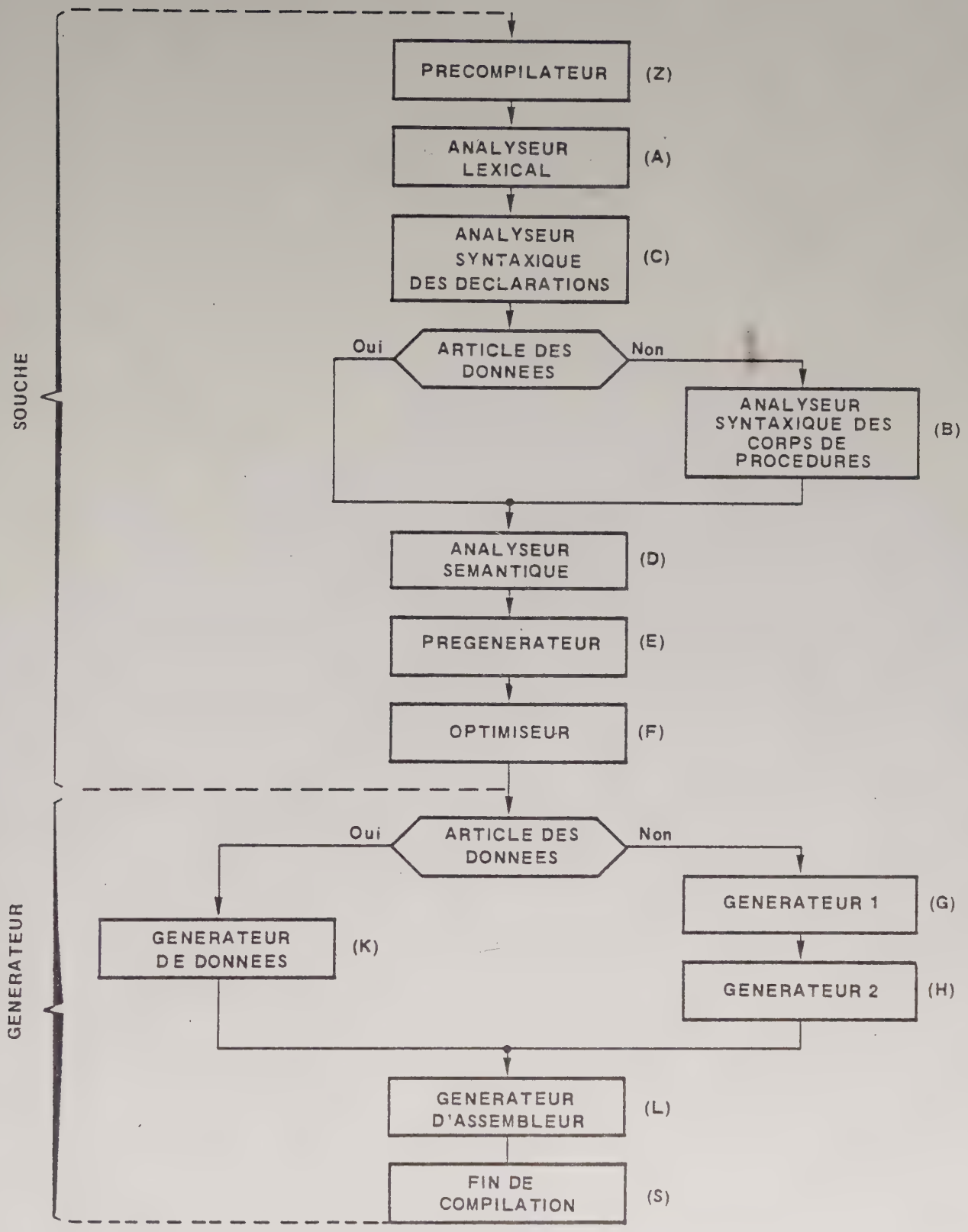


FIG. 2-1 - STRUCTURE D'UN COMPILATEUR LTR

NOM du FICHIER **Fichier des Erreurs** Nb. CAR/ART

LABEL

Application :	
Chaine : UT :	Réf : 21 -
Dessin de fichier	Date

Numéro d'ordre	Passe	Procédure	Nom du programme Machine cible Machine de compilation	Unité de compilation	Date de création de la fiche suiveuse	Date de correction	Numéro du rapport d'anomalie	Niveau de détection	Niveau d'anomalie	Type d'anomalie	Type de correction
00 01 02 03	04 05 06 07	08 09 10 11	12 13 14 15	16 17 18 19	20 21 22 23	24 25 26 27	28 29 30 31	32 33 34 35	36 37 38 39	40 41 42 43	44 45 46 47	48 49 50 51	52 53 54 55	56 57 58 59
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Numéro fiche suiveuse I numérique ou E	complément éventuel	Numéro de correction	Libre								
60 61 62 63	64 65 66 67	68 69 70 71	72 73 74 75	76 77 78 79	80 81 82 83	84 85 86 87	88 89 90 91	92 93 94 95	96 97 98 99		
15	16	17	18	19	20	21	22	23	24		

00 01 02 03	04 05 06 07	08 09 10 11	12 13 14 15	16 17 18 19	20 21 22 23	24 25 26 27	28 29 30 31	32 33 34 35	36 37 38 39	40 41 42 43	44 45 46 47	48 49 50 51	52 53 54 55	56 57 58 59
25	26	27	28	29	30	31	32	33	34	35	36	37	38	39

60 61 62 63	64 65 66 67	68 69 70 71	72 73 74 75	76 77 78 79	80 81 82 83	84 85 86 87	88 89 90 91	92 93 94 95	96 97 98 99
40	41	42	43	44	45	46	47	48	49

00 01 02 03	04 05 06 07	08 09 10 11	12 13 14 15	16 17 18 19	20 21 22 23	24 25 26 27	28 29 30 31	32 33 34 35	36 37 38 39	40 41 42 43	44 45 46 47	48 49 50 51	52 53 54 55	56 57 58 59
50	51	52	53	54	55	56	57	58	59	60	61	62	63	64

60 61 62 63	64 65 66 67	68 69 70 71	72 73 74 75	76 77 78 79	80 81 82 83	84 85 86 87	88 89 90 91	92 93 94 95	96 97 98 99
65	66	67	68	69	70	71	72	73	74

FIG. 2-2 - SCHEMA D'ENREGISTREMENT D'UN FICHIER D'ERREURS

A - CALCUL

- A000 Cas non prévus (code à éviter)
- A100 Opérande incorrect dans un calcul
- A200 Parenthésage incorrect dans un calcul
- A300 Opérateur incorrect dans une expression de calcul
- A400 Erreur d'unité, de conversion, de type
- A500 Calcul manquant
- A600 Expression entièrement incorrecte

B - LOGIQUE

- B000 Cas non prévus (code à éviter)
- B100 Opérande incorrect dans une expression logique
- B200 Code ne pouvant pas être atteint (par ex. à cause d'une condition incorrecte)
- B300 Tests sur variables incorrectes
- B400 Test logique (ou condition) manquant
- B500 Trop (ou pas assez) d'instructions en boucle
- B600 Nombre d'itérations d'une boucle incorrect
- B700 Opérateur incorrect dans un test logique
- B800 Contrôle aux bornes incorrect
- B900 Instruction manquante
- B950 Instruction en trop

C - ENTREES/SORTIES

- C000 Cas non prévus (code à éviter)
- C100 Lecture sur fichier d'entrée incorrect
- C200 Donnée écrite suivant spécification de format incorrecte (écriture rejetée)
- C300 Ecriture de donnée de format incorrect
- C400 Sortie manquante ou incomplète
- C500 Champ de sortie de taille incorrecte
- C600 Données lue dans un format incorrect
- C700 Trop (ou pas assez) de données lues
- C800 Positionnement erroné dans le fichier
- C900 Fichier non ouvert

D - MANIPULATION DES DONNEES

- D000 Cas non prévus (code à éviter)
- D100
- D200 Variable utilisée avant initialisation
- D210 Variable utilisée avant allocation de mémoire
- D300 Initialisation mal faite ou mal placée
- D400 Variable référencée avec un mauvais nom
- D500 Erreur d'indication

E - INTERFACES

- E000 Cas non prévus (code à éviter)
- E100 Appel à un sous programme non fait (ou mal placé)
- E200 Erreur d'interface (Ex : passage de paramètre incorrect)
- E210 Paramètre (nombre - type - taille) incorrect
- E220 Erreur de système d'unité
- E230 Incorrection de nombre, type, taille, unité lors d'appels de fonctions du langage
- E240 Une procédure modifie une valeur qu'elle ne devrait pas altérer
- E300 Sortie de sous programme manquante (ou mal placée)

F - DEFINITION DE DONNEES

- F000 Cas non prévus (code à éviter)
- F100 Donnée incorrectement définie ou dimensionnée
- F200 Donnée référencée en dehors des bornes
- F300 Donnée non déclarée
- F400 Déclaration mal placée
- F500 Déclaration multiple

G - BASE DE DONNEES

- G000 Cas non prévus (code à éviter)
- G100 Donnée non initialisée en base de données
- G200 Donnée mal initialisée
- G300 Unité incorrecte

H - SYNTAXE

- H000 Cas non prévu (code à éviter)
- H100 Séparateur ou mot-clé oublié
- H200 Séparateurs ou mots-clés trop nombreux
- H300 Imbrication incorrecte de phrases conditionnelles (IF)
- H500 Phrase de CAS incorrectement construite
- H600 Expression syntaxiquement incorrecte
- H700 Séquencement d'instructions incorrect
- H800 Oubli de fermeture de commentaire

O - OPTIMISATION

- O000 Cas non prévu (code à éviter)
- O100 Optimisation code source
- O200 Optimisation généré

X - OUTILS UTILISES EN PRODUCTION

- X000 Outils imparfaits

Y - OUTILS D'AIDE A LA MISE AU POINT

- Y000 Outils de mise au point

Z - ERREUR DANS UN LANGAGE SECONDAIRE

- Z000 Cas non prévu (code à éviter)
- Z100 Assembleur

FIG. 2-3 - CODIFICATION DES ANOMALIES

CODAGE DE LA PHASE D'ORIGINE D'ANOMALIE

Il s'agit de la phase qui a été à l'origine de l'anomalie

Code	Phase d'origine de l'anomalie
S	Cahier des CHARGES
F	Analyse fonctionnelle
O	Analyse organique
P	Programmation
M	Méthodologie de mise en oeuvre incohérente (commande incorrecte, module mal placé, etc...) en phase d'intégration.
C	Correction précédente en phase de maintenance

CODAGE DE LA PHASE DE DETECTION D'ANOMALIE

Il s'agit de la phase au cours de laquelle l'anomalie a été détectée.

Code	Phase de détection
R	INSPECTION
C	COMPILATION
T	TEST
I	INTEGRATION DU PRODUIT
M	MAINTENANCE

FIG. 2-4 - CODAGE DES PHASES D'ORIGINE ET DE DETECTION

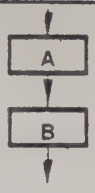
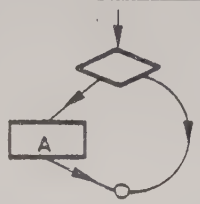
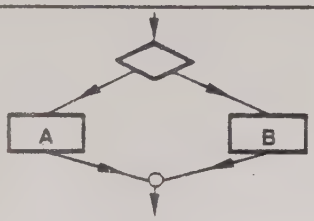
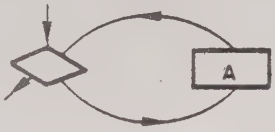
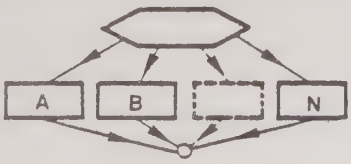
CONSTRUCTION	GRAPHE	CONTRIBUTION CYCLOMATIQUE	EXPRESSION ASSOCIEE
SOMMET INITIAL		1	P = 1
SEQUENCE		0	P = P (A) × P (B)
IF....THEN...		1	P = P (A) + 1
IF....THEN....ELSE		1	P = P (A) + P (B)
DO....WHILE....		1	P = P(A) + 1
DO....UNTIL....		1	$P = P(A) + P(A)^2$
CASE....OF N		N - 1	P = P (A) + P (B) ....+ P (N)

FIG. 2-5 - CONTRIBUTION CYCLOMATIQUES ET EXPRESSIONS ASSOCIEES AU CALCUL DU NOMBRE DE PARCOURS

CHAPITRE 3

1 ANALYSE DES DONNEES D'ERREURS

1.1 Analyse préliminaire des données d'erreurs

1.1.1 analyse des niveaux de détection

1.2 Répartition des sous types d'erreurs

1.3 Répartition des types d'erreurs

1.4 Relations entre types d'erreurs et unités fonctionnelles

1.4.1 la souche

1.4.2 le générateur

1.5 Répartition des erreurs par origines

1.6 Relations erreurs et origines d'erreurs

1.6.1 au niveau des sous-types d'erreurs

1.6.2 au niveau des types d'erreurs

1.7 Conclusions

CHAPITRE 3

Nous avons déjà mentionné que nous étions en présence de deux familles d'erreurs, la première, classique en fiabilité du logiciel, relève les erreurs de Calcul, Logique, Interface,... et correspond aux codes d'erreurs A, B, C, D, E, F, G, H. La seconde, définie pour couvrir l'ensemble des erreurs du compilateur LTR, mentionne des erreurs d'Outils de mise au point, d'Optimisation, des erreurs commises dans un autre langage, et commence par les codes O, X, Y, Z. Cette famille représente environ 25% des données d'erreurs.

Afin de respecter l'existence et la signification de ces deux familles de types d'erreurs dans une étude de fiabilité, seules les classes relatives à des définitions d'erreurs classiques en fiabilité du logiciel seront "actives" dans les traitements statistiques. Les deux familles d'erreurs seront conservées lorsque l'analyse se fera au niveau des sous-types, c'est à dire les classes spécifiques au compilateur participeront au traitement à titre d'individus supplémentaires <LeMoTa>. Par contre elles seront systématiquement éliminées pour tous traitements au niveau des types d'erreurs afin de pouvoir comparer les résultats avec ceux d'autres études semblables de fiabilité du logiciel publiés dans la littérature.

Ainsi, les données d'erreurs seront souvent analysées en deux étapes:

- la première étudie le fichier de données dans sa quasi-totalité en distinguant les deux familles d'erreurs dans les traitements;
- la seconde analyse un fichier de données réduit pour mieux tester les résultats.

La figure 3.1 fournit la répartition des 1224 erreurs en fonction des unités fonctionnelles. Aucune relation de proportionnalité apparait entre le nombre d'erreurs dans une procédure et le nombre de lignes de code écrites. Des analyses approfondies sont de toute évidence nécessaires.

Parmi les 1224 erreurs, 49 soit mal relevées soit mal codées n'ont pu être analysées. L'étude des 1175 relevés d'erreurs parfaitement identifiées sera faite en fonction des quatre attributs:

- l'unité fonctionnelle concernée,
- le type de l'erreur,
- la phase d'origine de l'erreur,
- la phase de détection de l'erreur.

1 ANALYSE DES DONNEES D'ERREURS

1.1 Analyse préliminaire des données d'erreurs

La figure 3.2 donne la répartition des 1175 sous-types d'erreurs et les proportions de chaque type d'erreurs. Beaucoup de sous-types

d'erreurs définis avant l'étude ont été peu ou jamais utilisés.

Il est fréquent en statistique d'éliminer les enregistrements suspects et peu représentatifs, lesquels correspondent par définition à des valeurs remarquables. Ce seuil d'individus au delà duquel une classe est jugée représentative constitue une valeur arbitraire, dépendant néanmoins du nombre total d'individus à analyser et de la qualité des données estimée à partir du processus de collecte. Ce seuil a été fixée à dix erreurs.

Les sous types d'erreurs de la figure 3.2 entourés d'un rond seront conservés dans la suite des traitements. Pour la même raison la phase d'origine des erreurs Méthodologie a été éliminée, figure 3.3.

1.1.1 analyse des niveaux de détection

La figure 3.3 montre également que la plupart des erreurs sont détectées en phase de Test des procédures du compilateur LTR, et la figure 3.4 indique que la quasi-totalité des erreurs sont détectées en phase de Test si l'on excepte les unités fonctionnelles A, C, G, Z.

La faible proportion d'erreurs d'unités A et C détectées en phase de Test s'explique par la détection de plusieurs erreurs en phase de Revue. Une analyse des corrections associées à ces erreurs prouve qu'il ne s'agit pas de correction au sens classique d'erreurs de fiabilité mais de modifications de programme liées à des changements de spécifications.

D'autre part la forte proportion d'erreurs de l'unité G détectées en phase d'Intégration s'explique par l'historique de développement du générateur. L'unité fonctionnelle G, la première du générateur à avoir été conçue, a été intégrée à la souche du compilateur sans avoir été testée. La détection de plusieurs erreurs en phase d'Intégration mise en évidence à la figure 3.4 traduit la difficulté à déterminer si les erreurs sont détectées en phase de Test ou d'Intégration. La fusion des deux phases de détection d'erreurs permet de dépasser cette ambiguité.

La phase de Test des procédures a ainsi assuré la détection de la majorité des erreurs des procédures du compilateur LTR.

Ces erreurs seront caractérisées par les trois attributs:

- l'unité fonctionnelle concernée,
- les catégories d'erreurs excédant dix unités,
- toutes les phases d'origine, exceptée la Méthodologie.

Ce nouvel ensemble de données d'erreurs possède 1085 enregistrements; soit plus de 92% des données initiales.

1.2 Répartition des sous-types d'erreurs

La figure 3.5 donne la répartition des sous-types d'erreurs par unités fonctionnelles. A l'exception des sous-types suivants:

- Calcul manquant,
- Variable référencée avec un mauvais nom,
- Outils imparfait,

représentant 20% du fichier de données d'erreurs, mais environ 3% des erreurs classiques en fiabilité du logiciel, on observe que les sous types d'erreurs sont fréquents parmi les unités fonctionnelles de la souche ou parmi celles du générateur.

Une Analyse Factorielle des Correspondances effectuée sur les données de la figure 3.5 permet de représenter, figure 3.6, la projection des unités fonctionnelles sur le premier plan factoriel.

> Différentes interprétations peuvent être énoncées à la lecture de la figure 3.6, la première remarque peut être que les projections sur le premier axe factoriel des unités fonctionnelles de la souche et du générateur sont respectivement proches et éloignées du point origine, la seconde remarque: hormis l'unité fonctionnelle B les autres sont relativement voisines.

Une caractérisation plus précise des unités fonctionnelles doit être entreprise.

Le découpage d'une taxinomie de types d'erreurs en sous-types est souvent assez ambigu et ne correspond pas toujours à une définition significative des sous-types. Il existe souvent une très forte dépendance entre deux sous-types d'erreurs d'un même type. La propriété d'équivalence distributionnelle vérifiée par les distances employées par l'Analyse Factorielle des Correspondances est particulièrement importante pour ce genre de données car elle permet d'avoir des résultats stables si l'on agrège les sous-types d'erreurs par types.

1.3 Répartition des types d'erreurs

La figure 3.7 liste les types d'erreurs analysés, donne le pourcentage d'erreurs par type et indique à titre de comparaison des pourcentages semblables de types d'erreurs de deux autres études de fiabilité. Les erreurs de Logique sont les plus nombreuses, les pourcentages d'erreurs d'Interface et de Manipulation de données sont plus faibles et représentent respectivement 19% et 15% des erreurs du compilateur LTR, enfin les erreurs de Calcul sont généralement peu fréquentes et couvrent 6% de l'échantillon d'erreurs analysées.

1.4 Relations entre types d'erreurs et unités fonctionnelles

Les figures 3.8 et 3.9 donnent respectivement les répartitions des types d'erreurs par unité fonctionnelle et des unités fonctionnelles par type d'erreurs.

Une Analyse Factorielle des Correspondances effectuée sur ces données montre aux figures 3.10 et 3.11 les projections sur le premier plan factoriel des unités fonctionnelles et des types d'erreurs. La figure 3.10 distingue graphiquement les procédures de la souche de celles du générateur. Plus précisément, l'Analyse Factorielle des Correspondances autorise une interprétation simultanée des projections des deux ensembles sur les axes factoriels. En rapprochant les figures correspondantes, à l'exception des erreurs de Logique qui affectent l'ensemble des unités fonctionnelles du compilateur, les erreurs d'Interface et d'E/S se rencontrent parmi les

unités fonctionnelles issues de la souche et les erreurs de Calcul, de Définition de données, de Manipulation de données parmi celles du générateur.

Les relations entre les différentes unités fonctionnelles du compilateur LTR et les types d'erreurs rencontrés sont précisées à la figure 3.12 et s'expliquent surtout par les fonctionnalités et les modes de développement de la souche et du générateur.

1.4.1 la souche

Le précédent chapitre a montré que la souche du compilateur était une extension d'une "souche" d'un autre compilateur LTR IRIS 50. Les nouvelles fonctions ont été implémentées par adjonction de codage et les nouvelles interfaces, inhérentes par exemple à la création des unités fonctionnelles E et F, n'ont jamais été globalement spécifiées, résolues et ont donc souvent évoluées. La fonction de la souche -- délivrer des tables et des textes pour n'importe quelle machine cible-- pose de surcroît des problèmes d'interface et d'E/S que n'éprouvent pas les concepteurs du générateur.

1.4.2 le générateur

La fonction du générateur --traduire un ensemble de valeurs numériques transmises par la souche en Assembleur de la machine concernée à l'aide d'automates, intermédiaires entre les valeurs numériques et le programme décodé; il y a par exemple un automate pour l'instruction IF qui reconnaitra cette instruction parmi l'ensemble des valeurs numériques et produira le code Assembleur correspondant (load, store, ...)-- l'oblige à traiter de nombreuses données qu'il doit définir, calculer, manipuler. Des erreurs de ces types en fortes proportions étaient inévitables lors de la réalisation du générateur du compilateur LTR.

Plus précisément, la figure 3.12 associe les erreurs de Manipulation de données aux unités fonctionnelles G, K; les erreurs de Définitions de données aux unités fonctionnelles K et L et assigne à l'unité fonctionnelle H des erreurs de Calcul.

- L'unité fonctionnelle G, la première unité du générateur, lit le texte intermédiaire transmis par la souche et produit, à l'aide de modèles de génération de code, du pseudo code assembleur. De même l'unité fonctionnelle K génère les données en code assembleur au moyen surtout de modèles d'assignation. Des modifications du texte intermédiaire transmis par la souche, liées à des extensions de celle-ci par exemple, provoquent de nombreuses nouvelles manipulations de données des unités G et K et inévitablement des erreurs de ce type. Par une astuce de programmation les modèles de génération de l'unité fonctionnelle G étaient paramétrés (dans le champ instruction, la valeur 01 est définie par load, de même 03 dans le deuxième champ signifie une instruction indexée notée *, etc). La mauvaise interprétation des modèles de génération, plus fréquente lorsque le travail se fait sur des données numériques plutôt que sur un texte, explique

la forte proportion d'erreurs de Définition de données de l'unité fonctionnelle K.
- L'unité fonctionnelle H optimise l'utilisation de la mémoire. De nombreux calculs d'adresses sont effectués et l'état des registres est fréquemment précisé, ces traitements génèrent des erreurs de Calcul.
- Enfin l'unité fonctionnelle L traduit le pseudo code en code Assembleur au moyen de tables de traduction. La détection de plusieurs erreurs de Définition de données s'explique par l'apparition d'erreurs dans l'emploi des tables de traduction, d'autre part le pseudo code possède toute l'information assembleur et les manipulations de données sont peu fréquentes.

1.5 Répartition des erreurs par origines

La figure 3.3 montre que 66% de l'ensemble des erreurs du compilateur LTR ont pour origine de mauvaises spécifications et près de 30% un codage incorrect. Ces pourcentages sont semblables à ceux trouvés dans les deux autres études de fiabilité déjà mentionnées; Thayer obtient 62% d'erreurs de spécifications et 38% d'erreurs de codage, <Th78>, et Motley respectivement 64% et 36%, <MoBr77>.

1.6 Relations entre erreurs et origine d'erreurs

1.6.1 au niveau sous-types d'erreurs

La figure 3.13 fournit le profil des sous-types d'erreurs en fonction de leurs phases d'origine. L'examen de cette figure montre des analogies parmi certains sous-types d'erreurs, ceux par exemple de codes A100, A500, A600, B400, D400, H700 n'ont jamais été créés au cours des phases de spécifications globales (fonctionnelles ou cahier des charges) mais lors des spécifications organiques ou du codage; soit lors des ultimes phases de réalisation d'un produit logiciel. Par contre les erreurs de code B900, F300 sont surtout intervenues pendant les spécifications. L'existence de relations entre les sous-types d'erreurs et leurs origines apparait à la lecture de la figure 3.13; elle est confirmée de facon analytique par l'application du test du khi-2.

Une Analyse Factorielle des Correspondances sur les données de la figure 3.13 ne précise pas convenablement les relations entre la phase de développement d'un produit logiciel et les sous-types d'erreurs les plus probables comme le montre la figure 3.14.

Le niveau d'analyse est ici encore probablement trop fin. Pour les mêmes raisons qu'au paragraphe ci dessus, l'analyse des relations entre les erreurs du compilateur LTR et leur origine doit être conduite au niveau des types d'erreurs.

1.6.2 au niveau des types d'erreurs

La propriété d'équivalence distributionnelle vérifiée par les distances de l'Analyse Factorielle des Correspondances permet de conserver au niveau des types d'erreurs l'existence de relations entre les sous-types

et les phases de développement établie précédemment.

La lecture de la figure 3.15 permet de constater que lors de la réalisation du cahier des charges du compilateur LTR, plus de 50% des erreurs sont des erreurs d'Interface; de même, près de 50% des erreurs de Calcul sont commises lors du codage des procédures. Nous obtenons ainsi par simple lecture les premières relations entre les phases de développement d'un produit logiciel et le type d'erreurs le plus probable.

Ces relations sont affinées par l'application d'une Analyse Factorielle des Correspondances aux données de la figure 3.15. Une interprétation simultanée des projections des types d'erreurs et de leurs origines sur les premiers plans factoriels est possible. Les figures 3.16 et 3.17 confirment bien entendu que les erreurs d'Interface ont pour origine le Cahier des charges. Ces figures précisent toutes les relations déductibles de la figure 3.15 et montrent qu'à chaque type d'erreurs n'existe qu'une origine probable. L'ensemble des relations entre les différentes phases de production d'un logiciel et les types d'erreurs les plus probables est précisé à la figure 3.18. Ces relations sont en accord avec l'association que l'on peut faire par expérience entre chaque phase de développement et le type d'erreurs le plus fréquent dans cette phase.

Les seules relations publiées dans la littérature entre les différents types d'erreurs et leurs origines sont recopiées au tableau 1 où les spécifications regroupent les trois phases de spécification utilisées dans notre étude <Th78>.

	Codage	Spécifications
Calcul	10	90
Logique	12	88
E/S	76	24
Manip. données	75	25
Interface	17	83
Déf. données	38	62

Tableau 1

A l'exception des erreurs de Calcul, les données de Thayer confirment les relations de la figure 3.18, de façon indirecte pour les erreurs de E/S.

1.7 Conclusions

L'emploi de méthodes statistiques élaborées a permis de mettre en évidence deux résultats de nature différente:

La fonctionnalité de la souche et du générateur, voire leurs conditions de développement, rendent compte de la localisation dans le programme des types d'erreurs créées. Ce résultat, propre au logiciel analysé, montre que les données d'erreurs recueillies sont bien homogènes.

Les liaisons entre le type d'erreurs et son origine dans la phase du cycle de vie constituent des relations relativement indépendantes du logiciel analysé, et largement établies empiriquement. La vérification de ces liaisons sur les données du compilateur LTR indique que celles-ci sont assez fiables.

Ces deux résultats sont importants, ils constituent une validation de la qualité des données analysées.

Souche

Passes	Z	A	B	C	D	E	F
Lignes source	1436	4390	2875	4711	12744	11958	3224
Anomalies	21	249	38	151	67	28	27

Générateur

Passes	G	H	K	L
Lignes source	18861	4952	4688	5738
Anomalies	276	108	90	169

FIG. 3-1 - DESCRIPTION DES PROGRAMMES - VOLUME DES ANOMALIES

	000	100	200	300	400	500	600	700	800	900	SOMME	POURCENTAGE
A	0	(22)	0	4	4	(10)	(12)	0	0	0	52	4,43
B	0	(14)	4	(18)	(94)	0	(12)	9	9	(138)	298	25,36
C	0	0	0	5	3	3	0	0	2	0	13	1,11
D	0	0	(19)	(88)	(13)	3	0	0	0	0	123	10,47
E	0	(23)	(134)	1	0	0	0	0	0	0	158	13,45
F	0	(64)	3	(80)	8	3	0	0	0	0	158	13,45
G	0	0	(12)	0	0	0	0	0	0	0	12	1,02
H	0	0	0	0	0	0	4	(14)	0	0	18	1,53
O	(57)	7	7	0	0	0	0	0	0	0	71	6,04
X	(171)	0	0	0	0	0	0	0	0	0	171	14,55
Y	(18)	0	0	0	0	0	0	0	0	0	18	1,53
Z	(44)	5	(34)	0	0	0	0	0	0	0	83	7,06
											1175	100,00

(entourés d'un ◯ figurent les sous-types d'erreurs conservés dans le traitement statistique)

FIG. 3-2 - REPARTITION DES SOUS-TYPES D'ERREURS

	S	F	O	P	C	M	
TOTAL	112	307	374	330	45	7	1175
%	9,53	26,13	31,83	28,08	3,83	0,59	99,99

REPARTITION DES DIFFERENTES PHASES ORIGINE D'ERREURS

	R	C	T	I	M	
TOTAL	116	25	893	122	19	1175
%	9,87	2,12	76	10,38	1,62	99,99

REPARTITION DES DIFFERENTES PHASES DETECTION D'ERREURS

FIG. 3-3

Phase de détection	Unités Fonctionnelles											
	A	B	C	D	E	F	G	H	K	L	Z	
revue	27.6	0.0	38.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	9.9
compilation	10.7	0.0	0.0	0.0	0.0	3.7	0.0	0.0	0.0	0.0	0.0	2.1
test	46.7	100.0	43.3	100.0	100.0	96.3	73.9	100.0	97.8	100.0	66.7	76.
intégration	14.7	0.0	5.7	0.0	0.0	0.0	26.1	0.0	2.2	0.0	33.3	10.3
maintenance	0.4	0.0	12.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.6
somme	19.1	3.2	12.0	5.6	1.2	2.3	23.5	9.2	7.7	14.3	1.8	100.0

FIG. 3-4 - REPARTITION DES ERREURS PAR PHASE DE DETECTION

	A	B	C	D	E	F	G	H	K	L	Z	SOMME
Opérande incorrecte dans une expression	0	1	1	3	0	0	0	13	0	4	0	22
Calcul manquant	2	0	3	0	0	0	4	0	0	1	0	10
Expression entièrerement incorrecte	0	0	0	0	0	0	1	8	3	0	0	12
Opérande incorrecte dans une expression logique	5	5	0	0	0	0	0	2	0	2	0	14
Test sur des variables incorrectes	1	0	0	0	1	0	6	3	0	7	0	18
Test logique manquant	24	0	19	3	1	0	24	7	3	2	11	94
Nombre d'itérations d'une boucle incorrect	2	7	2	0	1	0	0	0	0	0	0	12
Instruction manquante	5	0	9	10	6	2	3	23	29	50	1	138
Variable utilisée avant initialisation, allocation mémoire	0	0	2	0	0	0	0	0	6	11	0	19
Initialisation mal faite	6	2	18	1	0	0	36	1	12	12	0	88
Variable référencée avec un mauvais nom	7	0	0	0	0	0	6	0	0	0	0	13
Appel à un sous programme mal fait, mal placé	6	2	4	3	0	0	0	1	5	1	1	23
Erreur d'interface	49	5	17	8	5	9	14	4	11	7	1	130
Donnée incorrectement définie, dimensionnée	16	0	15	1	0	2	4	1	5	15	3	62
Donnée non déclarée	0	0	9	0	0	0	0	11	15	45	0	80
Donnée initialisée avec une valeur incorrecte	0	0	5	3	0	0	0	3	0	1	0	12
Séquencement d'instructions incorrect	10	0	0	0	0	0	3	0	0	1	0	14
Ø000	0	0	1	0	0	0	44	12	0	0	0	57
Outils imparfaits	51	0	0	32	0	0	85	2	1	0	0	171
Outils de mise au point	0	0	0	0	0	0	4	14	0	0	0	18
Z000	32	0	5	1	0	1	5	0	0	0	0	44
Z200	0	0	0	0	0	0	34	0	0	0	0	34
SOMME	216	22	110	65	14	14	273	105	90	159	17	1085

FIG. 3-5 - REPARTITION DES TYPES D'ERREURS PAR UNITE FONCTIONNELLE

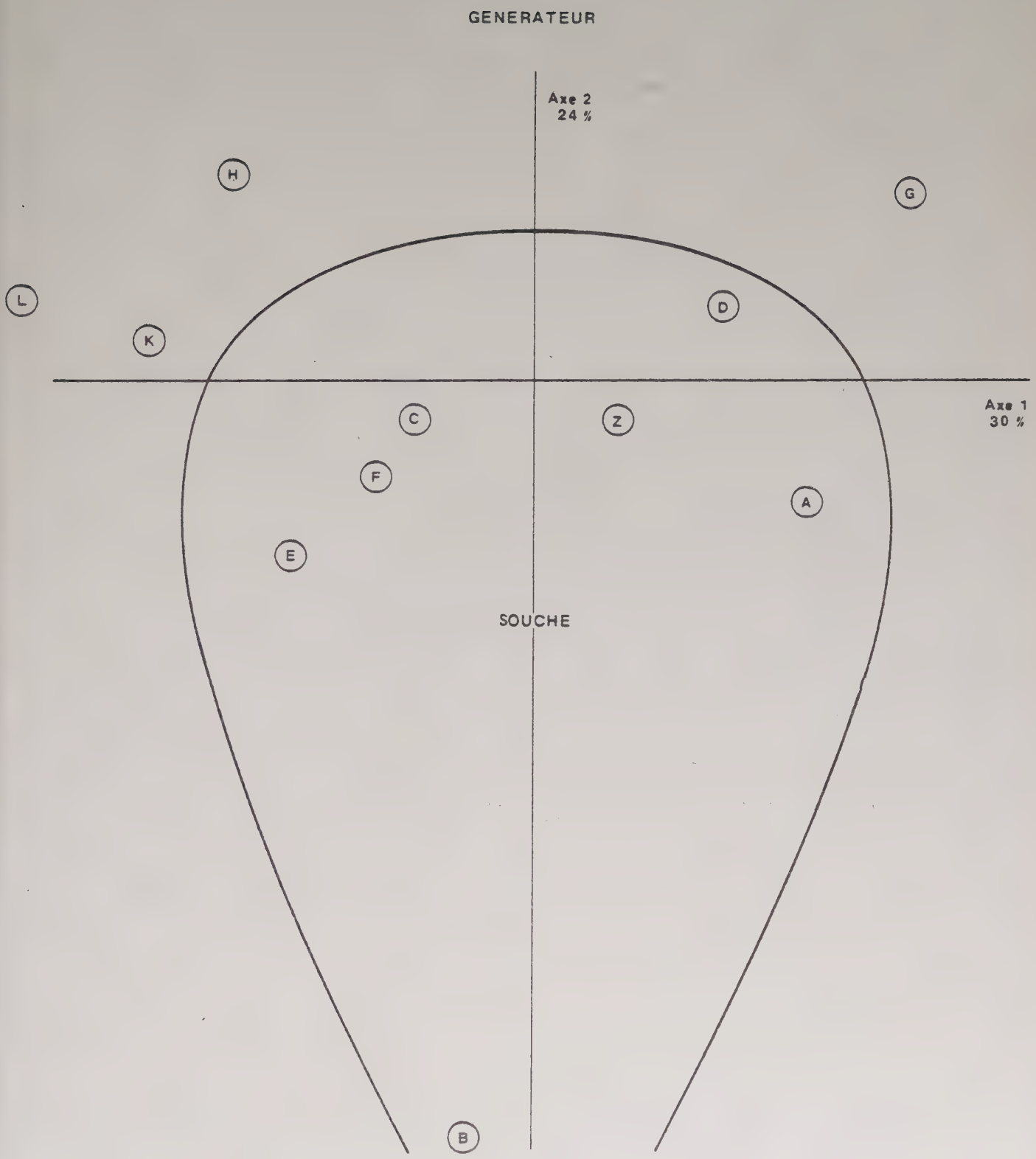


FIG. 3-6 - PROJECTION DES UNITES FONCTIONNELLES SUR LE 1er PLAN FACTORIEL

Types d'erreurs	Cette étude	[Th78]	[MoBr77]
Calcul	6	9	9
Logique	38	26	26
Entrée/Sortie	2	14	16
Manipulation de données	15	18	18
Interface	19	16	17
Définition des données	19	3	1
Base de données	1	7	4
Autres	-	7	9
Somme	100	100	100

FIG. 3-7 - REPARTITION DES ERREURS PAR TYPES

TYPE D'ERREUR	A	B	C	D	E	F	G	H	K	L	Z
Calcul	3.5	5.3	3.1	1.4	0.0	6.7	5.8	27.5	3.4	3.6	0.0
Logique	34.8	57.9	27.9	40.6	64.3	20.0	34.6	45.0	36.0	37.9	70.6
E/S	0.7	13.2	5.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Manipulation de données	9.2	5.3	15.5	3.1	0.0	0.0	42.3	1.3	20.2	14.2	0.0
Interface	39.0	18.4	18.6	34.4	35.7	60.0	13.5	6.3	18.0	5.9	11.8
Définition de données	12.8	0.0	25.6	3.1	0.0	13.3	3.8	16.3	22.5	37.9	17.6
Base de données	0.0	0.0	3.9	9.4	0.0	0.0	0.0	3.8	0.0	0.6	0.0
	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0

FIG. 3-8 - REPARTITION DES TYPES D'ERREURS POUR CHAQUE UNITE FONCTIONNELLE

Unités / Fonct. / rreurs	A	B	C	D	E	F	G	H	K	L	Z	%
alcul	9.6	3.84	7.69	5.77	0	1.96	11.54	42.31	5.77	11.54	0	100.00
ogique	13.31	5.80	11.94	4.78	3.07	1.02	11.26	12.28	10.92	21.50	4.09	100.00
/S	7.69	38.46	53.84	0	0	0	0	0	0	0	0	100.00
Manip. données	10.57	1.62	16.26	0.81	0	0	35.77	0.81	14.63	19.51	0	100.00
nterface	35.71	4.54	14.28	7.14	3.25	5.84	9.1	3.25	10.38	5.19	1.3	100.00
Déf. données	11.26	0	16.9	0.7	0	1.40	2.80	8.45	14.08	42.25	2.11	100.00
Base de données	7.69	0	50.00	11.54	0	0	0	15.38	0	15.38	0	100.00

FIG. 3-9 - REPARTITION DES UNITES FONCTIONNELLES POUR CHAQUE TYPE D'ERREUR

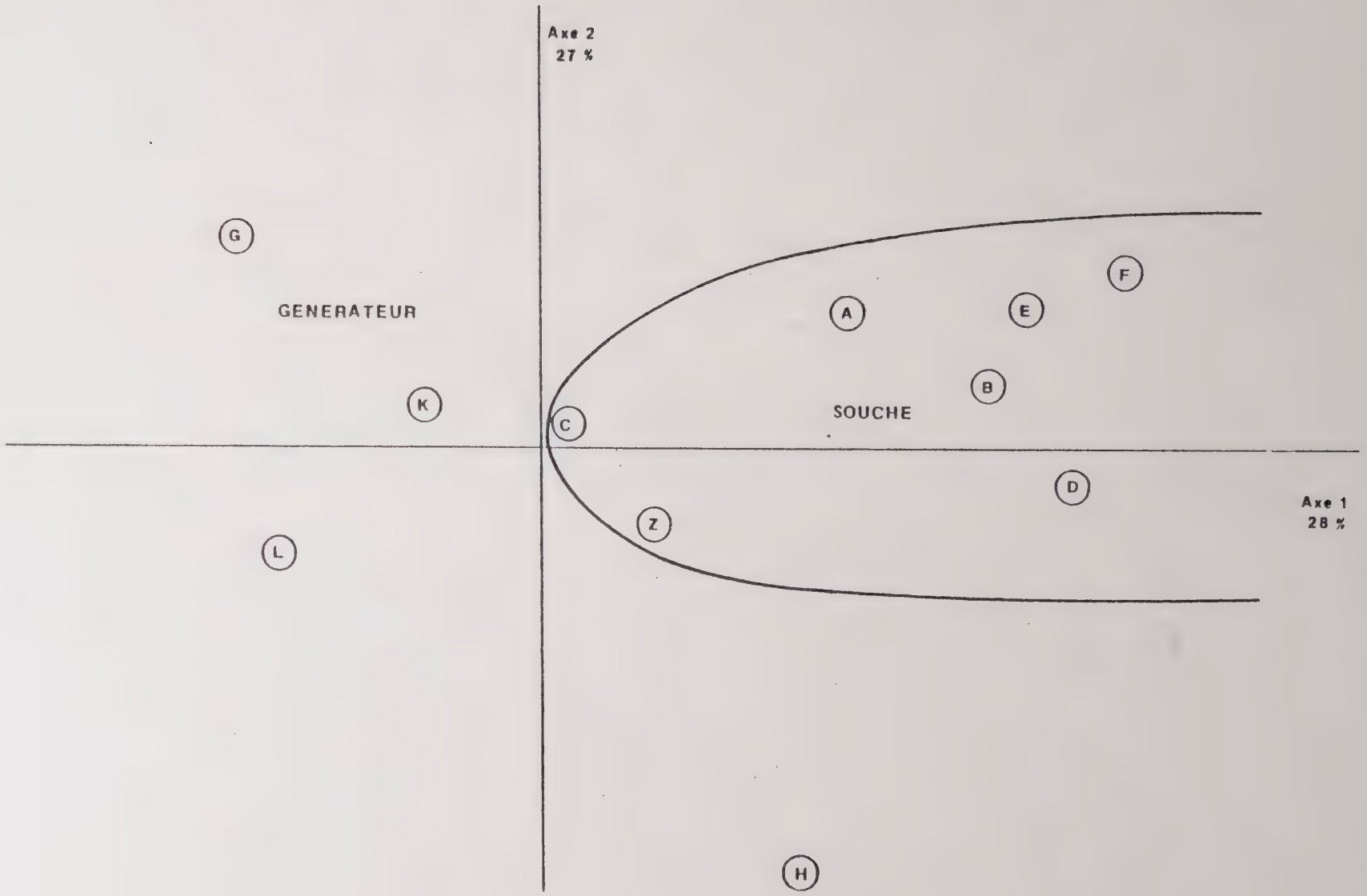


FIG. 3-10 - PROJECTION DES UNITES FONCTIONNELLES SUR LE 1er PLAN FACTORIEL

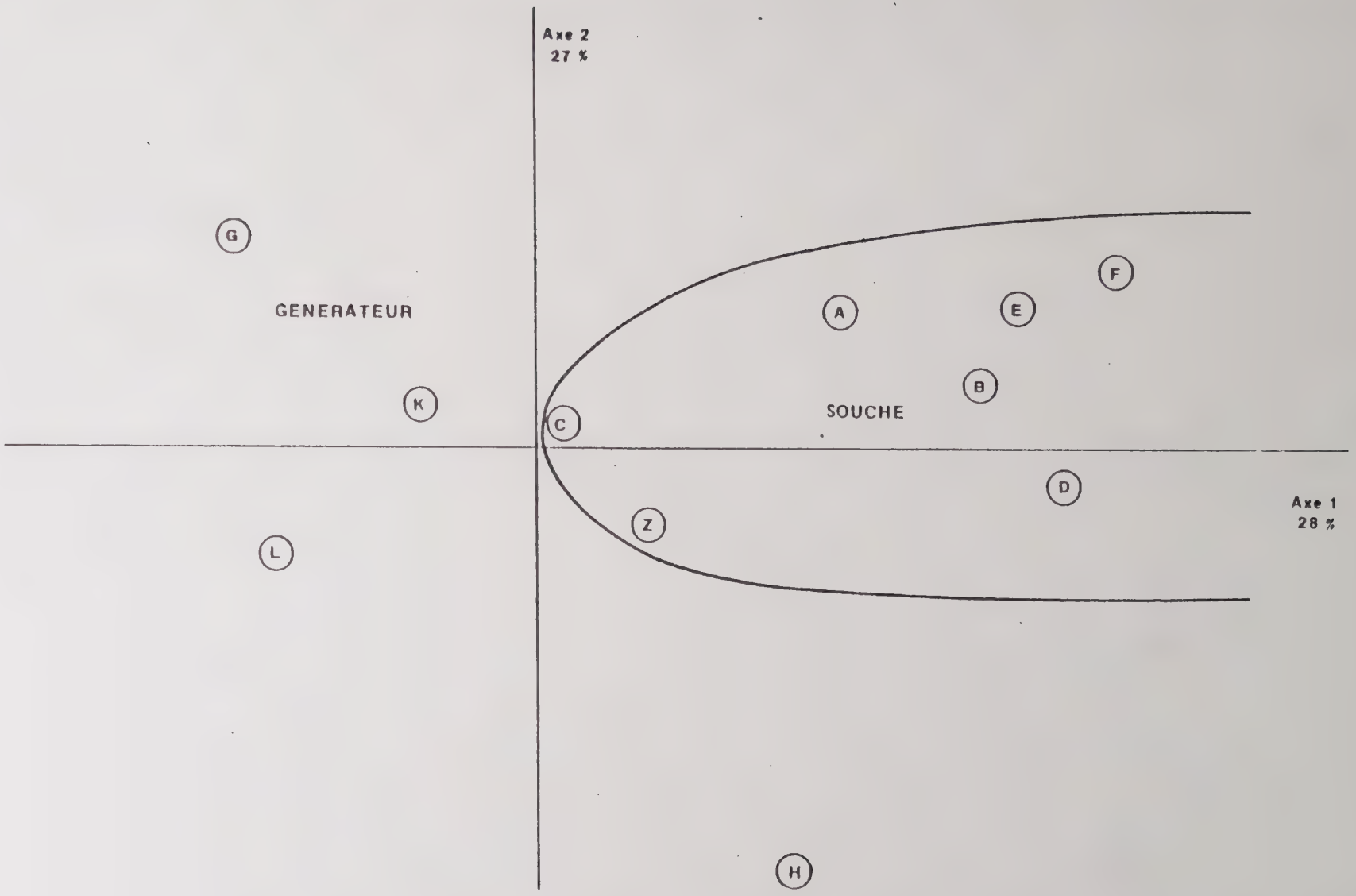


FIG. 3-10 - PROJECTION DES UNITES FONCTIONNELLES SUR LE 1er PLAN FACTORIEL

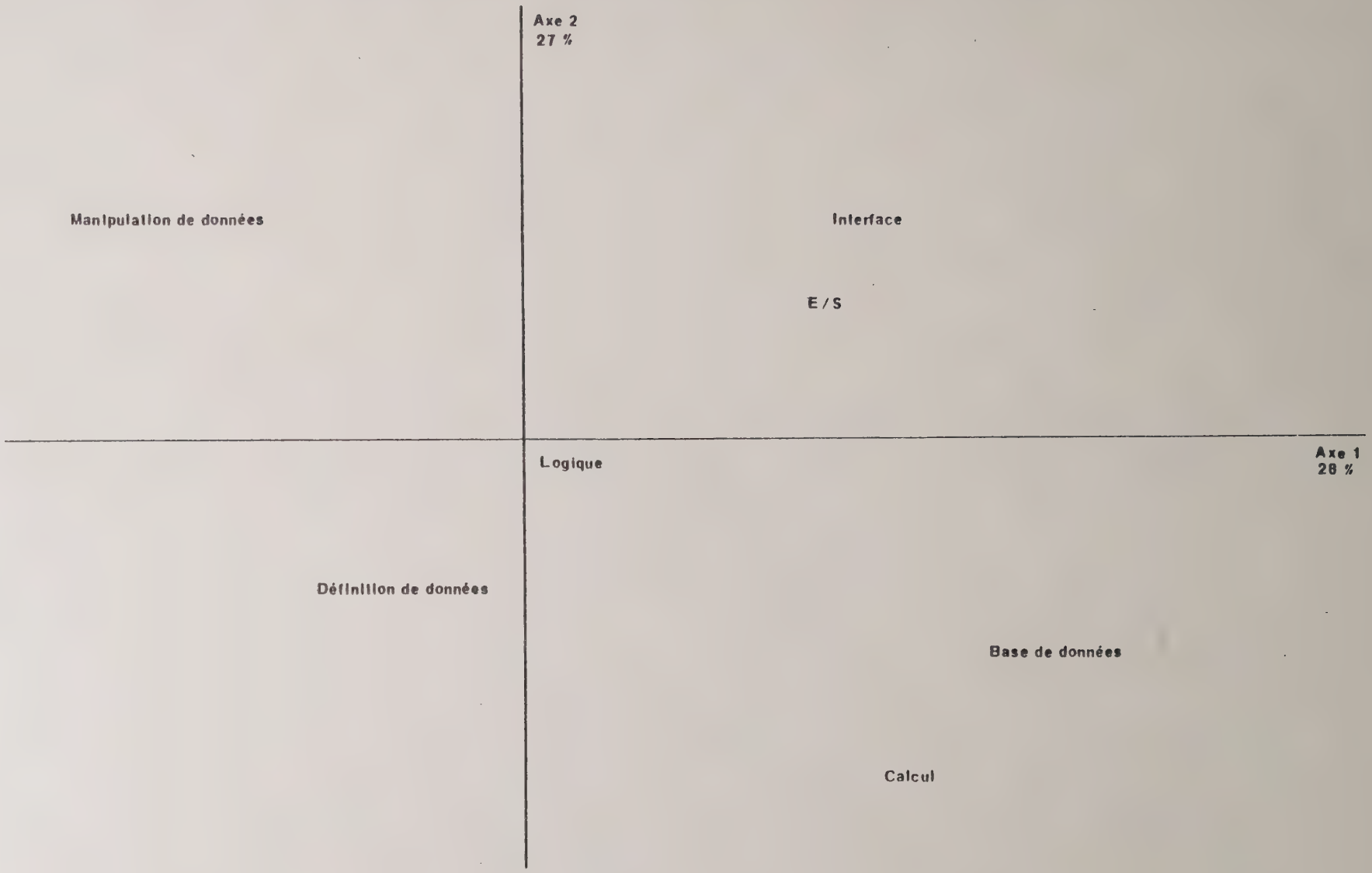


FIG. 3-11 - PROJECTION DES TYPES D'ERREUR

UNITE FONCTIONNELLE	TYPES D'ERREUR
Z	Logique
A	Interface, E/S
B	Interface, E/S
C	
D	Interface, E/S
E	Interface, E/S
F	Interface, E/S
G	Manipulation de données
H	Calcul
K	Manipulation de données Définition de données
L	Définition de données

FIG. 3-12 - RELATION ENTRE UNITES FONCTIONNELLES ET TYPE D'ERREUR

	A100	A500	A600	B100	B300	B400	B600	B900	D200	D300	D400	E100	E200	F100	F300	G200	H700	Ø000	X000	Y000	Z000	Z200	SOMME
S	0	0	0	0	0	0	0	18	0	2	0	1	41	14	2	0	0	3	30	0	0	1	112
F	0	0	1	0	5	5	4	52	10	7	0	10	27	22	48	2	0	2	78	0	4	11	288
O	7	3	7	9	6	52	0	52	7	41	0	7	33	14	28	5	6	28	48	0	0	5	358
P	12	6	2	5	5	33	7	9	2	37	13	5	26	10	1	5	8	24	13	16	40	15	294
C	3	1	2	0	2	4	1	7	0	1	0	0	3	2	1	0	0	0	2	2	0	2	33
	22	10	12	14	18	94	12	138	19	88	13	23	130	62	80	12	14	57	171	18	44	34	1085

	A100	A500	A600	B100	B300	B400	B600	B900	D200	D300	D400	E100	E200	F100	F300	G200	H700	Ø000	X000	Y000	Z000	Z200
S	0	0	0	0	0	0	0	13	0	2,3	0	4,3	31,5	22,6	2,5	0	0	5,3	17,5	0	0	2,9
F	0	0	8,3	0	27,7	5,3	33,3	37,7	52,6	7,9	0	43,4	20,7	35,5	60	16,6	0	3,5	45,6	0	9,1	32,3
O	31,8	30	58,3	64,3	33,3	55,3	0	37,7	36,8	46,6	0	30,4	25,4	22,6	35	41,6	42,8	49,1	28	0	0	14,7
P	54,5	60	16,6	35,7	27,7	35,1	58,3	6,5	10,5	42	100	21,7	20	16,1	1,2	41,6	57,1	42,1	7,6	88,8	90,9	44,1
C	13,7	10	16,6	0	11,1	4,3	8,3	5	0	1,1	0	0	2,3	3,2	1,2	0	0	0	1,2	11,1	0	5,8

FIG. 3-13 - REPARTITION DES SOUS-TYPES D'ERREUR PAR PHASES D'ORIGINE

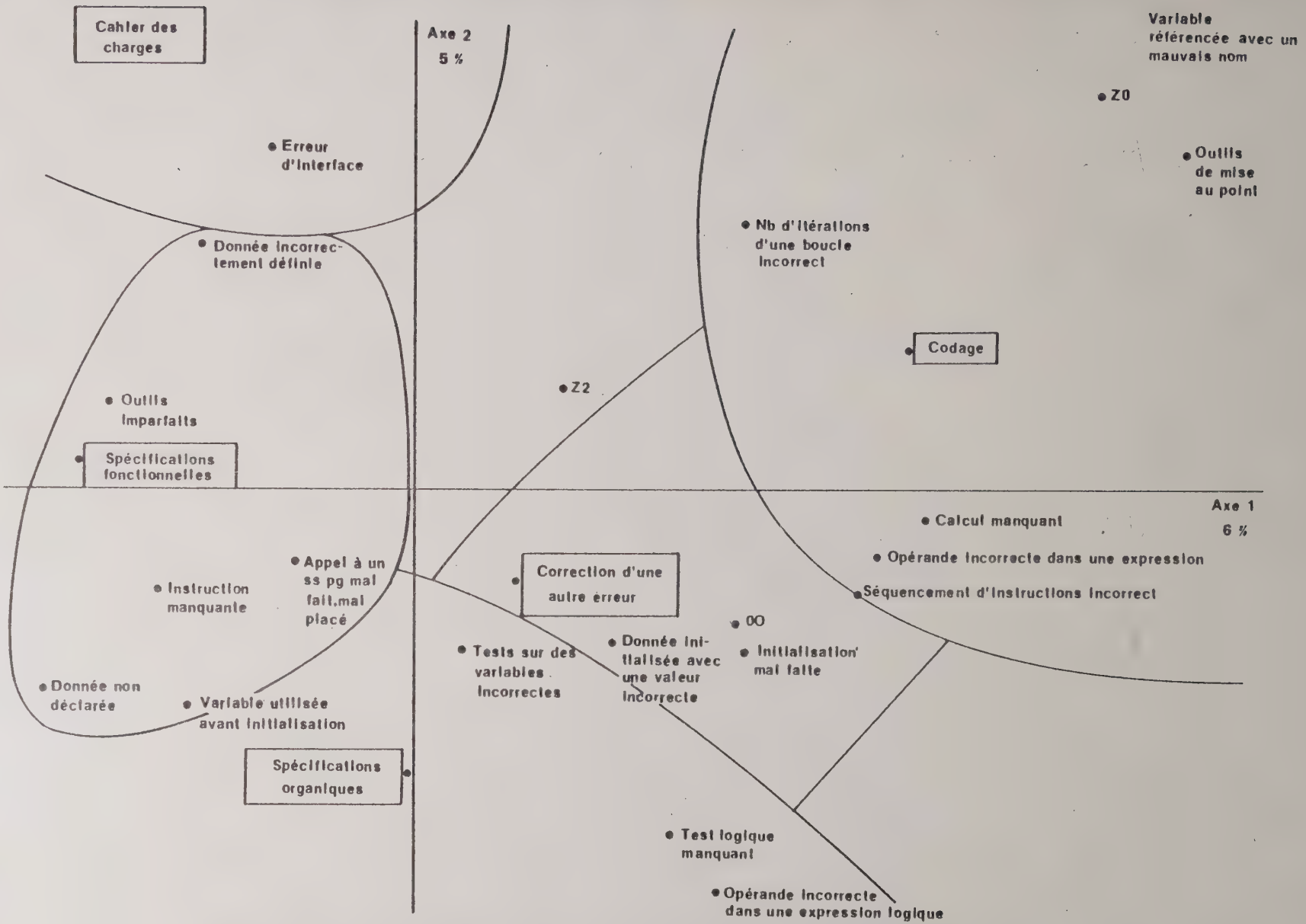


FIG. 3-14 - PROJECTION DES SOUS-TYPES D'ERREURS ET DE LEURS ORIGINES SUR LE 1er PLAN FACTIORIEL

CATEGORIE	CAHIER DES CHARGES	SPECIFICATION FONCTIONNELLE	SPECIFICATION ORGANIQUE	CODAGE	CORRECTION	NOMBRE D'ERREURS	POURCENTAGE
Calcul	0	2	19	24	7	52	6.3
Logique	18	70	129	81	14	312	38.0
E/S	0	3	0	4	6	13	1.6
Manipulation de données	2	17	49	54	1	123	15.0
Interface	42	38	40	31	3	154	18.7
Définition des données	16	73	46	16	5	156	19.0
Base de données	0	2	5	5	0	12	1.4
Total	78	205	288	215	36	822	100.0

FIG. 3-15 - REPARTITION DES TYPES D'ERREURS PAR NIVEAU D'ORIGINE

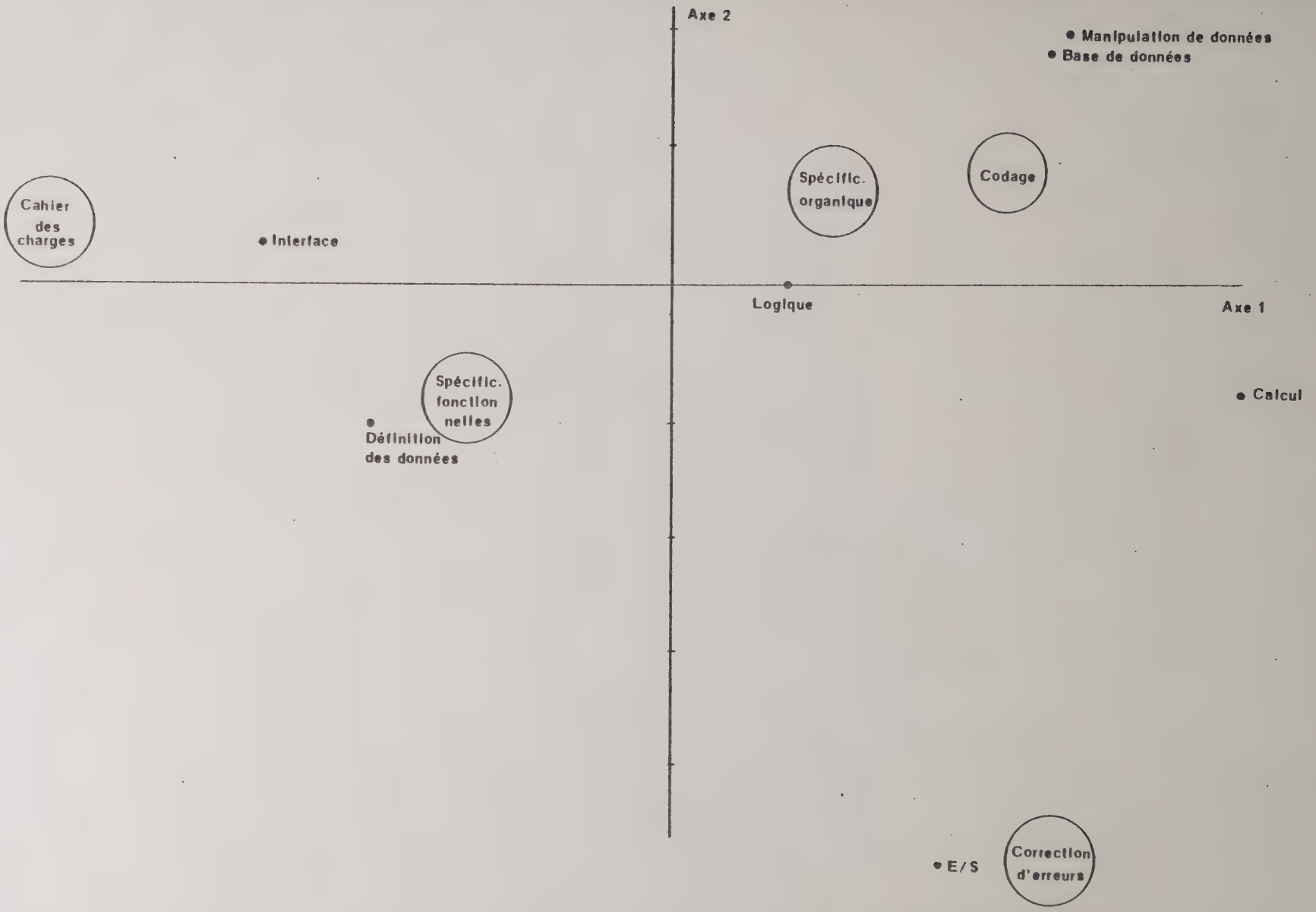


FIG. 3-16 - PROJECTION DES TYPES D'ERREURS ET DE LEURS ORIGINES SUR LE 1er PLAN FACTORIEL

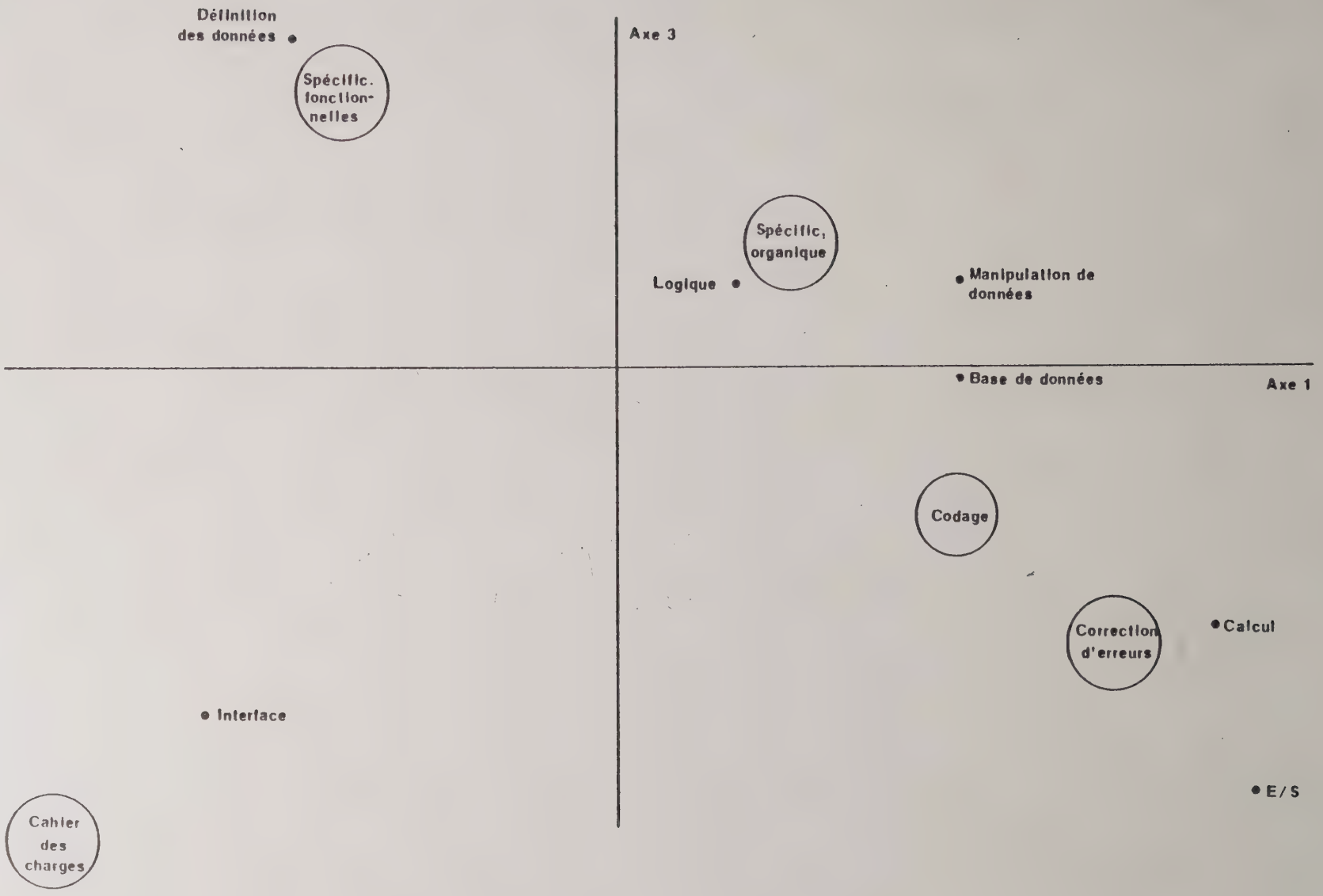


FIG. 3-17 - PROJECTION DES TYPES D'ERREURS ET DE LEURS ORIGINES SUR LE 2ème PLAN FACTORIEL

ORIGINES	TYPES D'ERREURS
Cahier des charges	Interface
Spécification fonctionnelle	Définition des données
Spécification organique	Logique
Codage	Manipulation de données, base de données, calcul
Corrections d'erreurs	E/S, calcul

FIG. 3-18 - RELATIONS ENTRE ORIGINES ET TYPES D'ERREURS

CHAPITRE 4

1 ANALYSE DES MESURES DE COMPLEXITE DU COMPILATEUR LTR

1.1 Etude des mesures de complexité

1.2 Définition des nouvelles mesures de complexité

1.2.1 introduction des nouvelles mesures textuelles

1.2.2 définitions des nouvelles mesures structurelles

1.2.3 analyses des nouvelles mesures de complexité

1.3 Etude de la complexité du compilateur LTR

1.4 Conclusions de l'étude de complexité du compilateur LTR

1.5 Validation de l'équation de l'estimateur de taille

2 ANALYSE DES MESURES DE COMPLEXITE D'UN LOGICIEL PASCAL

2.1 Définitions des mesures

2.2 Statistiques de base

2.3 Corrélations des mesures

2.4 Prédicteur de taille

3 CONCLUSIONS DE L'ETUDE COMPARATIVE DES RESULTATS

CHAPITRE 4

La complexité d'un logiciel ne se définit pas simplement et fait de toute évidence intervenir un grand nombre de facteurs. Pour quantifier ces différents facteurs, nous avons indiqué que plusieurs mesures statiques de programmes avaient été proposées par différents auteurs au cours des cinq dernières années. Selon le type de complexité qu'elles caractérisent ces mesures peuvent être partitionnées en deux classes:

- les mesures de complexité textuelle,
- les mesures de complexité structurelle.

Nous présenterons dans ce chapitre l'analyse des mesures de complexité et les expériences de validation de la principale relation de la Science du Logiciel. Nous présenterons les mêmes statistiques sur un autre logiciel afin de tester la similitude des résultats.

1 ANALYSE DES MESURES DE COMPLEXITE DU COMPILATEUR LTR

1.1 Etude des mesures de complexité

Le dictionnaire des dix mesures de complexité utilisées dans l'étude, et définies au Chapitre 2, est donné à la figure 4.1. Les principaux calculs élémentaires effectués dans le cas d'un fichier de mesures sont présentés à la figure 4.2 et les coefficients de corrélation entre toutes les mesures sont donnés à la figure 4.3.

Il apparait tout d'abord que les mesures de complexité textuelle sont fortement corrélées entre elles. Ces résultats sont cohérents avec la possibilité d'estimer le nombre total d'opérateurs (resp. d'opérandes) utilisés dans la procédure à partir du nombre d'opérateurs distincts (resp. d'opérandes distincts) mais aussi, et ceci est souvent ignoré, ces coefficients de corrélation confirment l'existence de relations linéaires entre le nombre d'opérateurs distincts et le nombre d'opérandes distincts au moyen de coefficients ne dépendant que du nombre d'opérandes distincts du programme écrit dans le langage de plus haut niveau (cf: définition du niveau de programme: CHAPITRE 1) du type de celle proposée par Halstead <Ha77>.

La figure 4.3 met aussi en évidence les fortes dépendances des nombres cyclomatiques avec les mesures de complexité textuelle. D'autres auteurs utilisent dans leur étude la variable effort censée caractériser la complexité textuelle et trouvent des coefficients de corrélation élevés entre le nombre cyclomatique et la variable effort, souvent voisins de 0.95 <CuSh79, Su81>.

Enfin, la figure 4.3 indique une relative indépendance entre les mesures d'atteignabilité, le nombre de chemins et les autres mesures de complexité. L'absence de références bibliographiques liée au non emploi de ces deux mesures dans la plupart des études traditionnelles empêche de confronter ces résultats avec ceux d'autres études.

La matrice de corrélation des mesures de complexité montre

néammoins la nécessité de définir de nouvelles mesures avec l'objectif de mieux distinguer les aspects textuels de la complexité des procédures de ses aspects structurels.

1.2 Définitions des nouvelles mesures de complexité

1.2.1 introduction des nouvelles mesures textuelles

La plupart des mesures classiques de la Science du Logiciel ont été introduites, il s'agit:

- N la taille,
- n le vocabulaire,
- V le volume.

Ces trois mesures caractérisent différemment la taille d'un programme, directement, à partir de son vocabulaire, du nombre de bits requis pour le codage.

- En l'effort normalisé

La mesure de l'effort, largement employée dans les études expérimentales de mesures de complexité, détermine en reprenant la terminologie d'Halstead : "le nombre de discriminations mentales élémentaires". Cette mesure est normalisée afin d'avoir une densité, jugée plus significative qu'un nombre trop dépendant de la taille des procédures, déjà largement prise en compte par les trois mesures précitées.

- D la difficulté

Cette quantité évoque la difficulté à coder "une unité élémentaire de discrimination mentale". Fitsos conclue son étude expérimentale <Fit80> en indiquant que le nombre d'opérateurs distincts d'un programme écrit en langage évolué est sensiblement constant et préconise l'utilisation moyenne des opérandes (U2=n2/N2) comme variable prédictrice de la difficulté. Cette dernière mesure et la mesure traditionnelle niveau de langage, dont les valeurs dépendent des échantillons de données, ont été introduites.

La complexité textuelle est définie à l'aide des sept nouvelles mesures citées.

1.2.2 définitions des nouvelles mesures structurelles

Les fortes corrélations entre les mesures de la complexité textuelle et les nombres cyclomatiques peuvent être interprétées comme un effet "taille" puisque plusieurs études ont montré que les paramètres du vocabulaire constituaient un bon estimateur de la taille d'un programme. Les nombres cyclomatiques ont été normalisés par la taille de la procédure, définie par N=N1+N2, afin d'éliminer cet effet puis, par souci de cohérence des définitions de mesures, cette normalisation s'est étendue aux autres mesures de complexité structurelle, l'atteignabilité et le nombre de chemins. Six densités de complexité structurelle sont définies.

Deux mesures supplémentaires ont été employées afin de mettre en

évidence la linéarité du graphe de contrôle et le degré d'imbrication des instructions de test contenues dans les programmes. Plus précisément il s'agit

- l la linéarité du graphe

Ce nombre, compris entre 0 et 1 et calculé à l'aide des nombres cocyclomatique et cocyclomatique étendu, vaut 0 si le graphe de contrôle et le graphe étendu sont équivalents (aucune linéarité dans le bloc de programme) et 1 si le bloc de programme est purement linéaire.

- d l'indépendance des tests

Cette quantité est obtenue en estimant les nombres minimum et maximum de chemins pour un nombre cyclomatique donné. Elle vaut 1 si tous les tests sont en séquence et 0 s'ils sont tous emboités. La situation du nombre de chemins mesuré entre ces deux valeurs extrêmes fournit une indication sur le degré d'imbrication des tests.

1.2.3 analyses des nouvelles mesures de complexité

Un nouvel ensemble de quinze mesures a été ainsi défini et analysé. Le dictionnaire des nouvelles mesures est donné à la figure 4.4 et leur définition à la figure 4.5. Les mêmes calculs élémentaires que précédemment sont présentés à la figure 4.6, la matrice de corrélation des nouvelles mesures à la figure 4.7.

La valeur -0.67 du coefficient de corrélation entre la mesure difficulté et l'inverse de l'utilisation moyenne des opérandes infirme la suggestion de Fitsos d'estimer correctement une mesure au moyen de l'autre, autrement dit le nombre d'opérateurs distincts d'un programme écrit en langage évolué ne peut être considéré comme constant sur les données du compilateur LTR.

Les nouveaux coefficients de corrélation indiquent que les mesures de complexité textuelle et les mesures de complexité structurelle normalisées sont devenues moins liées.

- Toutes les représentations graphiques d'une nouvelle mesure de complexité structurelle en fonction d'une mesure de complexité textuelle permettent d'observer l'indépendance relative de ces deux familles de mesures (figures 4.8, 4.9, 4.10).
- Cette indépendance est aussi présentée à la figure 4.11 obtenue après application d'une Analyse en Composantes Principales sur l'ensemble de toutes les procédures du compilateur LTR caractérisées par leurs quinze mesures de complexité, les projections des mesures textuelles et des mesures structurelles sur le premier plan factoriel sont nettement différenciées.
De plus, la complexité textuelle dont les mesures sont projetées essentiellement sur le premier axe caractérise davantage la complexité des procédures que la complexité structurelle normalisée dont les mesures sont portées par

le deuxième axe. Cette remarque est intéressante car ce rapport hiérarchique entre les deux familles de mesures de complexité apparaitra à nouveau au chapitre suivant.

L'échantillon de mesures est satisfaisant pour pouvoir caractériser la complexité de chaque procédure du compilateur LTR.

1.3 Etude de la complexité du compilateur LTR

A l'instar de l'analyse des données d'erreurs, l'étude de la complexité du compilateur LTR s'est avérée fructueuse au niveau des unités fonctionnelles.

La figure 4.12 donne les valeurs moyennes des quinze mesures de complexité par unité fonctionnelle. Cette figure, lue en distinguant la souche du générateur, montre que les valeurs moyennes des principales mesures: la taille, le vocabulaire et la difficulté, sont élevées dans le cas des unités fonctionnelles A, E, F vis à vis des valeurs obtenues pour les autres unités fonctionnelles de la souche et faibles dans le cas de l'unité fonctionnelle L par rapport aux valeurs calculées pour les trois autres unités fonctionnelles du générateur. La confirmation graphique de ces observations est apportée à la figure 4.13 par la position des unités fonctionnelles A, E, F, L.

Un simple calcul permet de vérifier que le nombre moyen de lignes de code écrites par procédures pour chaque unité fonctionnelle caractérise un aspect de la complexité des procédures qui peut être indépendant des aspects textuels et structurels relevés par l' échantillon de mesures. Les unités C et D ont en effet des mesures de complexité semblables alors que l'unité fonctionnelle D possède des procédures en moyenne presque trois fois plus longues (39 lignes de code par procédure pour l'unité C contre 92 pour D). Or les principales mesures statiques choisies identifient surtout un effet "taille", cet effet n'est donc pas précisément bien pris en compte par la mesure du nombre de lignes de code (LOC). L'exposé bibliographique prouve que la mesure LOC est une variable d'étude de programmes moins significative que le nombre cyclomatique ou mieux la variable effort proposée par Halstead (CHAPITRE 1). Une des raisons tient à sa coercibilité de l'écriture d'un programme, elle dépend de facteurs extrinsèques à la fonction, par exemple la densité de commentaires.

Une autre interprétation de la figure 4.13 est de toute évidence nécessaire.

- L'unité A a été analysée et développée par de nombreux réalisateurs, chacun apportant sa couche de conception et d'analyse et probablement de complexité.
- L'unité E a été spécifiée et codée par la personne initialement la moins formée refusant souvent d'avoir une approche globale des problèmes de programmation.
- L'unité F a été étudiée et programmée par une personne très compétente, connaissant le langage LTR, usant d'astuces de conception et de programmation, et augmentant de cette facon la complexité de l'unité fonctionnelle.
- L'historique de développement de la souche rend compte de la plus grande complexité des unités A, E, F, que l'on

peut estimer d'ailleurs artificielle. Il est probable que des méthodes de développement plus rigoureuses auraient permis de diminuer la complexité de ces trois unités fonctionnelles.

- La moindre complexité de l'unité fonctionnelle L ne s'explique guère par ses conditions de développement, elle fut concue et réalisée par la même personne que l'unité K. La fonction de l'unité L, générer le code assembleur à partir du pseudo-code et des tables de traduction, conduit à des phases de conception et de réalisation plus élémentaires confirmées par les valeurs plus faibles des mesures de complexité.

1.4 Conclusions de l'étude de complexité du compilateur LTR

Deux observations se dégagent de l'étude des mesures de complexité du compilateur LTR.

Tout d'abord les mesures de complexité textuelle et structurelle, malgré un domaine d'application différent, sont fortement liées par les concepts de complexité qu'elles définissent, ce qui justifie en partie qu'aucune d'entre elles ne soit parvenue à s'imposer parmi les mesures statiques du logiciel. Dans leur étude comparative de mesures de complexité, Baker et Zweben proposent de rechercher une nouvelle mesure de complexité à l'aide de la variable effort et du nombre cyclomatique <BaZw79>. Ils montrent que cette nouvelle mesure de complexité possèderait simultanément les qualités et les mérites des mesures textuelles et structurelles puisqu'ils ne s'exercent pas concurremment. Sans attendre la définition d'une telle mesure, Paige préconise l'association de la variable effort au nombre cyclomatique pour pondérer dès la compilation les chemins d'un programme et orienter les tests de facon appropriée <Pa80>.

L'obtention d'une mesure de programmes associant étroitement les aspects textuels aux aspects structurels constitue certainement une étape vers la définition de mesures statiques du logiciel plus universelles.

La seconde observation issue de l'étude des mesures de complexité est la mise en évidence de l'existence de deux familles distinctes d'unités fonctionnelles, celles de la souche et celles du générateur. Ce résultat déjà apparu lors de l'analyse des données d'erreurs est important car il confirme indirectement l'existence de relations entre les mesures de complexité et les données d'erreurs, c'est à dire entre complexité et fiabilité.

1.5 Validation de l'équation de l'estimateur de la taille

L'estimation de la longueur de programmes au moyen du vocabulaire a été étudiée plusieurs fois; citons sans être exhaustif la validation de Bohrer sur de courts programmes, celle d'Elshoff sur un important échantillon de programmes commerciaux de taille variable écrits en PL1, celles de Fitsos sur des programmes écrits en assembleur et en langage évolué <Boh75, Els76, Fit80>.

L'équation des variations de la longueur d'une procédure en

fonction du vocabulaire utilisé a été testée sur les données du compilateur LTR. Le résultat obtenu

$$\Delta N = 1.02 * \Delta Ne$$

vérifie exactement la relation proposée. Comme l'étude de Fitsos, l'analyse des données du compilateur indique graphiquement à la figure 4.14 que les procédures de petite taille ont un estimateur de la longueur excédant la taille réelle, ce rapport s'inversant pour les procédures de grande taille. Les procédures de petite taille manipulent un minimum de vocabulaire d'où une estimation optimiste de leur longueur dans ce cas, en revanche le vocabulaire disponible est limité et borne les valeurs de la longueur estimée.

2 ANALYSE DES MESURES DE COMPLEXITE D'UN LOGICIEL PASCAL

2.1 Définitions des mesures

L'échantillon de données analysé dans ce second paragraphe correspond à un ensemble de procédures PASCAL développées par une équipe de recherches à l'INRIA. Cet ensemble possède 1000 procédures dont la moitié provient d'un système de manipulation de programmes par des arbres de syntaxe abstraite (Mentor) <DGHu79>. Chaque procédure PASCAL est caractérisée par six mesures de complexité:

- les quatre mesures textuelles de base,
- le nombre cyclomatique,
- le nombre cocyclomatique.

Afin d'entreprendre des études comparatives significatives, la complexité des procédures du compilateur doit être définie par ces six mesures.

2.2 Statistiques de base

La figure 4.15 présente les statistiques de base des six mesures.

Les procédures du compilateur LTR ont des valeurs moyennes légèrement supérieures aux valeurs des procédures PASCAL, sans jamais toutefois dépasser le rapport deux.

L'écart type associé à chaque variable permet de déterminer un intervalle de confiance contenant la valeur moyenne avec une probabilité de 0.95. L'expression des bornes de cet intervalle est:

$$I = < m - 1.96*sigma/n**(1/2) \ ; \ m + 1.96*sigma/n**(1/2)>$$

où m est la valeur moyenne, sigma l'écart type, n la taille de l'échantillon et 1.96 la valeur absolue du seuil au delà duquel une variable gaussienne, centrée, réduite a une probabilité de 0.05 d'être située. Les plages de valeurs des intervalles de confiance sont dans les deux cas sensiblement voisines, relativement courtes et permettent, sur l'ensemble des procédures PASCAL, l'obtention d'une valeur plus précise de la moyenne de chaque variable.

2.3 Corrélations des mesures

La figure 4.16 fournit les deux matrices de corrélation.

Les valeurs des coefficients sont très voisines voire dans quelques cas identiques: les données issues de l'échantillon de procédures PASCAL confirment l'existence de fortes dépendances entre ces six mesures de complexité et en particulier elles n'infirment pas le résultat d'Halstead d'estimer les trois autres mesures textuelles de base à partir de la mesure du nombre d'opérandes distincts.

La confirmation graphique de ces corrélations est apportée aux figures 4.17, 4.18 obtenues après application d'une Analyse en Composantes Principales sur chaque échantillon de données.

- Le pourcentage d'information initiale expliquée par le premier plan factoriel est le même sur les deux échantillons de données et correspond à 93% dont environ 87% pour le premier axe.
- L'équation de cet axe factoriel en fonction des six mesures de base est identique, son expression est dans les deux cas:

$$Y1 = -.84*n1 -.97*N1 -.91*n2 -.96*N2 -.96*nu -.97*V +e$$

où e est une variable aléatoire résiduelle.

L'existence de liaisons entre toutes les mesures de complexité apparait sur les deux figures par les projections des mesures sur le premier axe factoriel, seul significatif, en une même région. La position marginale du nombre d'opérateurs distincts sur le premier plan factoriel traduit les moindres corrélations de cette mesure avec l'ensemble des autres variables.

2.4 Prédicteur de taille

L'équation des variations du prédicteur de taille d'une procédure connaissant son vocabulaire, largement vérifiée expérimentalement, l'a été aussi sur les données du compilateur LTR. L'échantillon de procédures PASCAL met parfaitement en évidence l'existence d'une relation linéaire entre la mesure de taille et son prédicteur: 84% de la variance initiale traduit un phénomène linéaire. Le coefficient de la pente de la droite est en revanche différent de 1. L'expression obtenue est:

$$\Delta N = 0.77 * \Delta Ne$$

c'est à dire, l'estimateur de taille et la taille elle-même ne varient plus dans le même rapport. La représentation graphique de l'estimateur de la taille en fonction de la taille est donnée à la figure 4.19.

3 CONCLUSIONS DE L'ETUDE COMPARATIVE DES RESULTATS

L'étude comparative des résultats statistiques obtenus est raisonnable car les deux échantillons de données sont significatifs, plus de 1000 procédures analysées dans chaque cas, et distincts.

La principale différence de nature fonctionnelle entre les deux groupes de procédures est évidemment l'analyse d'un compilateur dans un cas, d'un ensemble de procédures dont environ la moitié appartient à un système de manipulation de programmes et l'autre moitié est d'origines diverses. Quelques différences ou caractéristiques des deux échantillons de procédures, qui pourraient influer sur la complexité des procédures, peuvent être brièvement énumérées. Le compilateur est un programme de traduction de langages, lié à une machine cible et un langage d'entrée; sa réalisation correspond à une tâche parfaitement définie. Le compilateur est un produit industriel alors que les procédures PASCAL ont toutes pour origine la même équipe de recherches de l'INRIA. De plus, la fonction du système Mentor comme certificateur de programmes multi-langages (gestion de programmes, analyse des programmes, mesures, mise au point, documentation, ...), et sa réalité comme produit en perpétuelle évolution, ne permettent pas à ses réalisateurs de le concevoir dans le même esprit que celui qui a pu animer les réalisateurs du compilateur LTR. Cette remarque doit être prise avec la réserve que les procédures issues du système Mentor ne représentent qu'une partie de ce système et que cette partie constitue environ la moitié de l'échantillon de procédures PASCAL analysées.

Malgré ces différences importantes, de nature fonctionnelle ou méthodologique, les résultats de l'analyse statistique des mesures de complexité sont semblables, voire identiques. Une seule alternative paraît expliquer ce phénomène:

- soit l'on admet que les mesures de complexité sont sujettes à varier avec les projets logiciels étudiés, mais alors les différences des deux programmes se sont traduites de facon telle qu'elles n'influeraient plus sur les relations entre les mesures de complexité,
- soit les liaisons entre les mesures de complexité s'établissent indépendamment du logiciel étudié et l'on admet qu'elles possèdent une valeur intrinsèque.

Confortée d'ailleurs par une étude bibliographique, cette deuxième hypothèse est évidemment beaucoup plus probable. Dans ce cas, le résultat de l'étude comparative est doublement important. Il confirme le bien-fondé de caractériser intrinsèquement les mesures de complexité, et les unes par rapport aux autres. Il assure une portée certaine aux études quantitatives de prédiction de fiabilité du logiciel en montrant que le phénomène identifié par les mesures de complexité est indépendant du logiciel, en d'autres termes il justifie la recherche de liaisons quantitatives entre les deux attributs complexité et fiabilité, objet du chapitre suivant.

nb opérateurs distincts	v1	η_1
nb opérateurs	v2	N_1
nb opérandes distinctes	v3	η_2
nb opérandes	v4	N_2
nb cyclomatique	v5	v
nb cocyclomatique étendu	v6	V'
nb cocyclomatique	v7	V
nb de chemins	v8	p
atteignabilité étendue	v9	R'
atteignabilité	v10	R

FIG. 4-1 - DICTIONNAIRE DES MESURES

VARIABLES	MINIMUM	MAXIMUM	MOYENNE	ECART TYPE
n_1	0.00000	95.000	12.316	9.2243
N_1	0.00000	1106.0	70.826	115.66
n_2	0.00000	156.00	20.022	20.375
N_2	0.00000	805.00	55.705	87.908
ν	1.0000	137.00	7.0291	13.167
V'	0.00000	447.00	20.344	35.881
V	0.00000	209.00	11.408	21.458
P	1.0000	0.10000E+07	20532.	0.13383E+06
R'	0.00000	0.10000E+07	29939.	0.15965E+06
R	0.00000	0.10000E+07	25782.	0.14692E+06

FIG. 4-2 - STATISTIQUES DES MESURES DE COMPLEXITE

	η_1	N_1	η_2	N_2	ν	V'	V	P	R	R'
η_1	1.00									
N_1	0.74	1.00								
η_2	0.78	0.87	1.00							
N_2	0.69	0.98	0.89	1.00						
ν	0.76	0.88	0.74	0.82	1.00					
V'	0.70	0.97	0.83	0.94	0.85	1.00				
V	0.78	0.93	0.83	0.89	0.96	0.91	1.00			
P	0.40	0.45	0.44	0.45	0.39	0.42	0.49	1.00		
R	0.44	0.54	0.51	0.55	0.46	0.53	0.57	0.90	1.00	
R'	0.43	0.49	0.49	0.50	0.43	0.47	0.54	0.96	0.97	1.00

FIG. 4-3 - MATRICE DE CORRELATION DES MESURES DE COMPLEXITE

nb opérandes distinctes/nb opérandes	v1	U_2
taille	v2	N
vocabulaire	v3	η
volume	v4	V_{Ol}
niveau du langage	v5	λ
effort/taille	v6	E_N
difficulté	v7	D
nb cyclomatique/taille	v8	ν_N
nb cocyclomatique étendu/taille	v9	V'_N
nb cocyclomatique/taille	v10	V_N
atteignabilité étendue/taille	v11	R'_N
atteignabilité/taille	v12	R_N
nb chemins/taille	v13	P_N
linéarité du graphe	v14	l
indépendance des tests	v15	d

FIG. 4-4 - DICTIONNAIRE DES MESURES DERIVEES

NOM	DEFINITION	
v_2	η_2/N_2	inverse utilisation opérande
N	N_1+N_2	taille
η	$\eta_1+\eta_2$	vocabulaire
$V_{o\ell}$	$N \text{ Log}_2\eta$	volume
λ	$V*(\frac{2}{\eta_1}.\frac{\eta_2}{N_2})^2$	niveau langage
E_N	$V*(\frac{\eta_1}{2}\frac{N_2}{\eta_2}) * \frac{1}{N}$	effort normalisé
D	$\frac{\eta_1}{2} . \frac{N_2}{\eta_2}$	difficulté
ν_N	ν/N	nb cyclomatique normalisé
V'_N	V'/N	nb cocyclomatique étendu normalisé
V_N	V/N	nb cocyclomatique normalisé
R'_N	R'/N	atteignabilité étendue normalisée
R_N	R/N	atteignabilité normalisée
P_N	P/N	nb de chemins normalisé
ℓ	$(V'-V)/V'$	linéarité du graphe
d	$(P-\nu)/(2^{\nu-1}-\nu)$	indépendance des tests

FIG. 4-5 - DEFINITION DES MESURES DERIVEES

Variables	Moyennes	Ecart-Type	Minimum	Maximum
U_2	0.5653	0.2269	0.0894	1.0000
N	131.2777	205.0232	4.0000	1911.0000
n	33.5377	27.9901	4.0000	193.0000
Vol	771.9976	1408.2271	8.0000	13055.9097
λ	3.9075	4.0717	0.2986	53.1508
EN	92.1450	130.2791	2.0000	1181.3304
D	16.4537	19.1727	1.0000	165.7012
ν_N	0.0689	0.0504	0.0015	0.4933
V'_N	0.1497	0.0536	0.0000	0.4867
V_N	0.0682	0.0584	0.0000	0.4867
R'_N	52.9900	294.1250	0.0258	3236.2427
R_N	44.8941	261.7825	0.0015	3236.2427
PN	34.6707	236.4758	0.0015	3236.2427
l	0.5217	0.3494	0.0000	1.0000
d	0.1610	0.3379	0.0000	1.0000

FIG 4-6. - STATISTIQUES DE BASE SUR LES MESURES DE COMPLEXITE DERIVEES

	U_2	N	η	Vol	λ	EN	D	ν_N	V'_N	V_N	R'_N	R_N	P_N	l	d
U_2	1.00														
N	-0.58	1.00													
η	-0.59	0.87	1.00												
Vol	-0.54	1.00	0.87	1.00											
λ	0.54	-0.26	-0.31	-0.25	1.00										
EN	-0.62	0.93	0.86	0.93	-0.35	1.00									
D	-0.67	0.92	0.84	0.90	-0.39	0.99	1.00								
ν_N	0.46	-0.15	-0.21	-0.13	0.08	-0.12	-0.14	1.00							
V'_N	-0.09	0.13	0.21	0.13	-0.04	0.15	0.15	-0.02	1.00						
V_N	-0.29	0.24	0.36	0.23	-0.36	0.32	0.34	0.36	0.43	1.00					
R'_N	-0.22	0.40	0.42	0.40	-0.12	0.43	0.42	-0.02	0.09	0.17	1.00				
R_N	-0.21	0.39	0.41	0.39	-0.12	0.43	0.42	-0.02	0.08	0.16	0.96	1.00			
P_N	-0.18	0.35	0.37	+0 .35	-0.10	0.39	0.37	-0.01	0.07	0.14	0.97	0.95	1.00		
l	0.18	-0.17	-0.23	-0.16	0.32	-0.22	-0.24	-0.45	0.18	-0.74	-0.11	-0.11	-0.10	1.00	
d	-0.13	-0.04	0.06	-0.05	-0.11	-0.01	0.01	-0.08	0.12	0.24	-0.07	-0.07	-0.06	-0.19	1.00

FIG 4-7. - MATRICE DE CORRELATION SUR LES MESURES DERIVEES

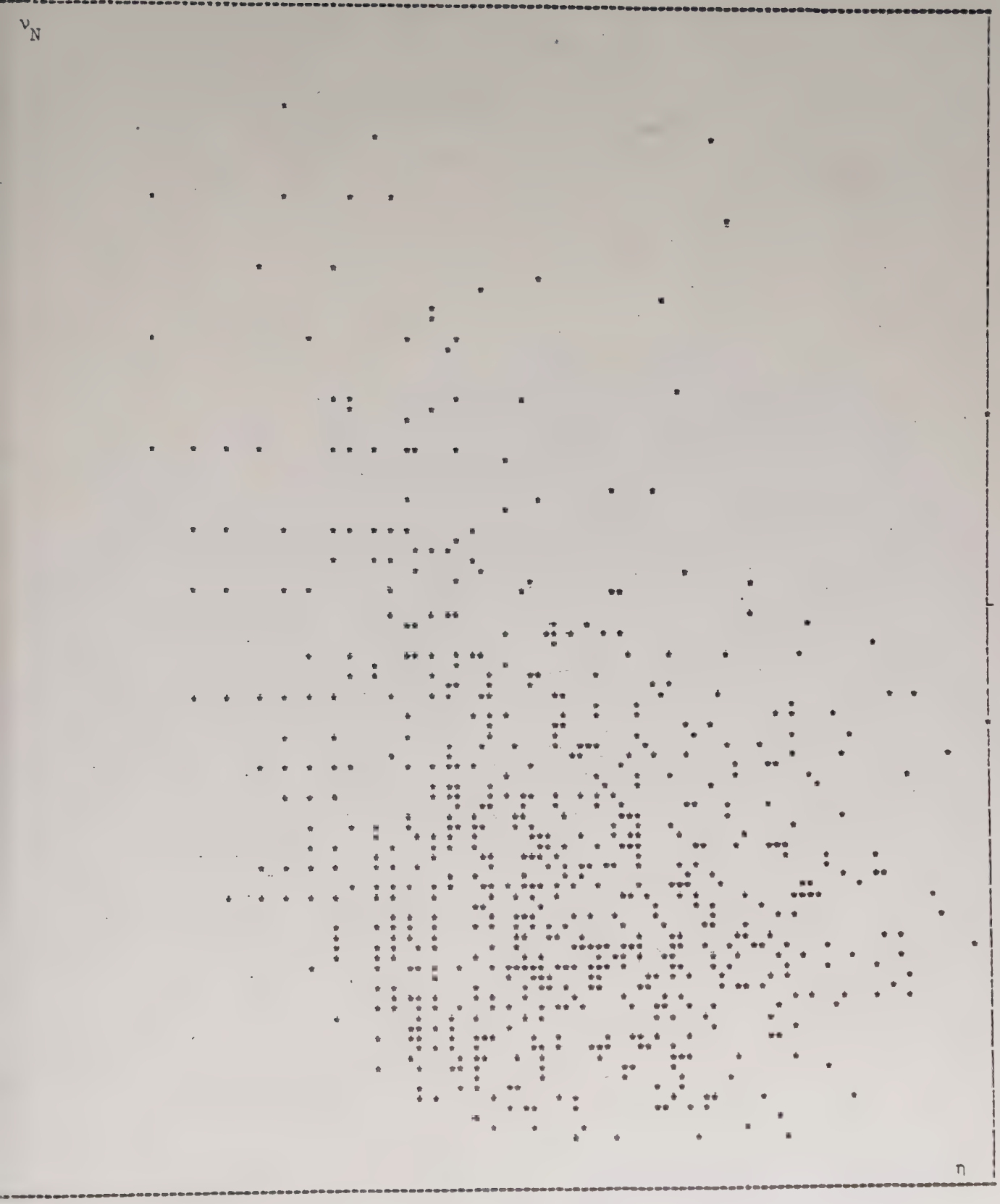


FIG. 4-8 - REPRESENTATION GRAPHIQUE DU NOMBRE CYCLOMATIQUE NORMALISE EN FONCTION DU VOCABULAIRE

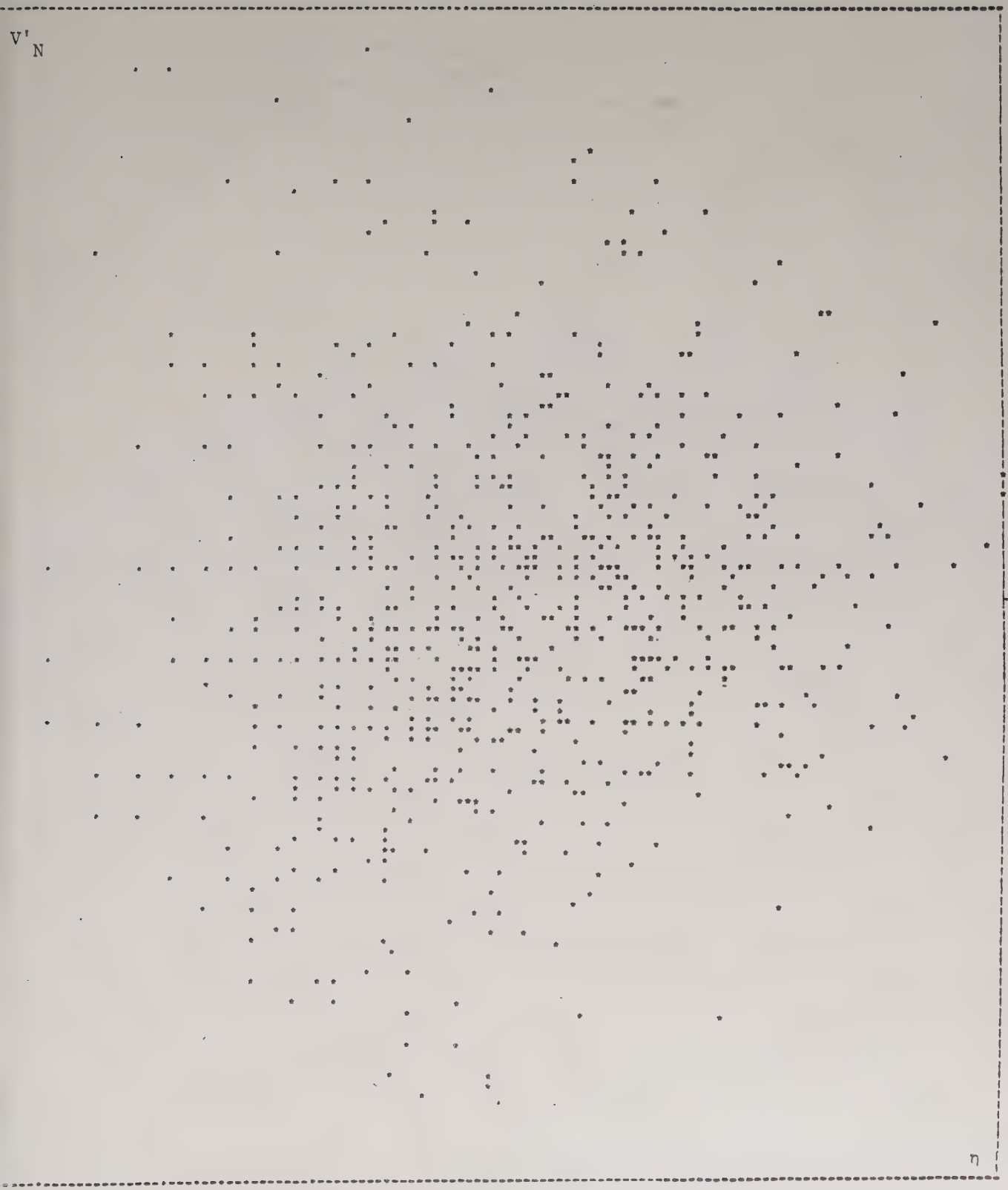


FIG. 4-9 - REPRESENTATION GRAPHIQUE DU NOMBRE COCYCLOMATIQUE ETENDU NORMALISE EN FONCTION DU VOCABULAIRE

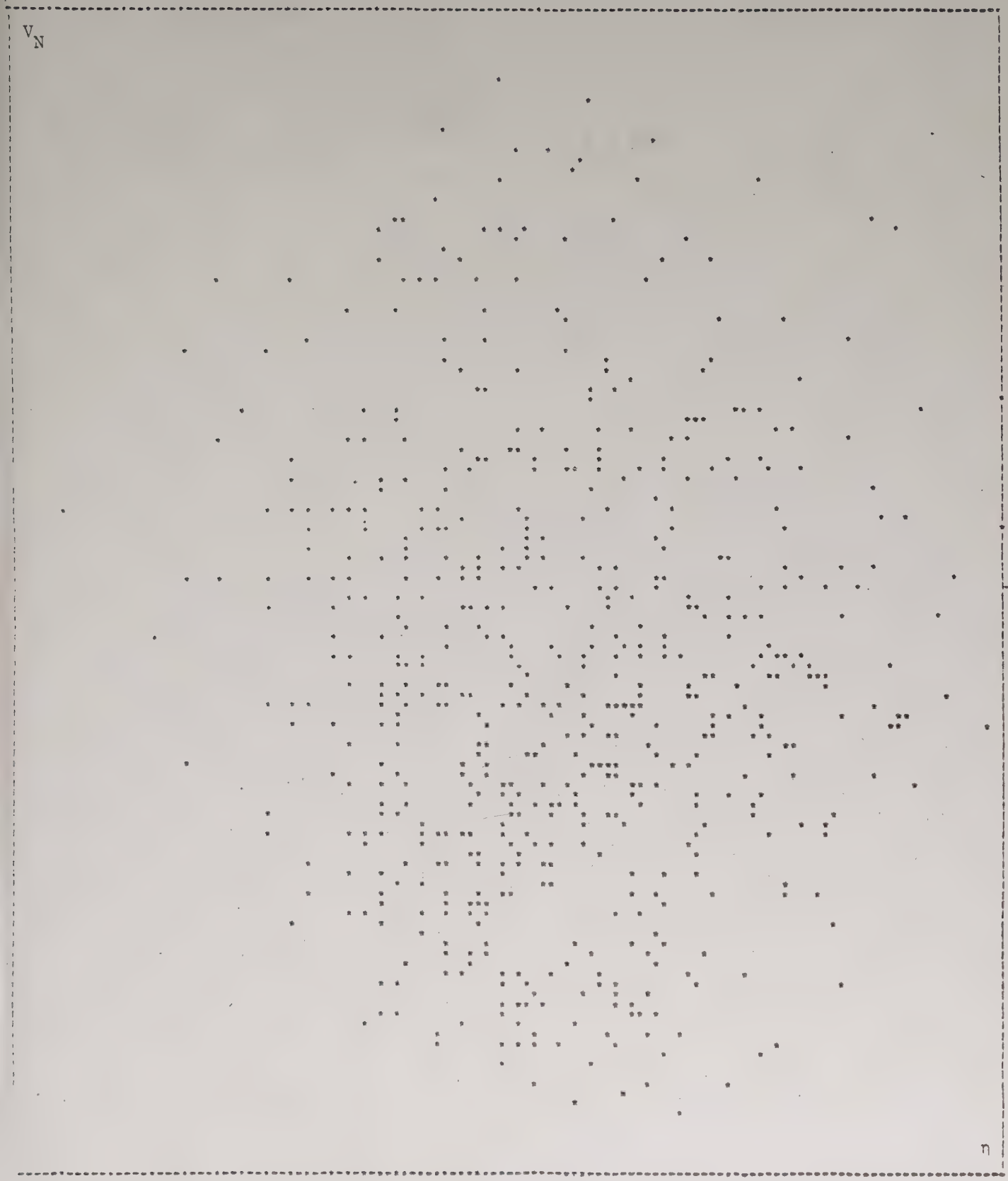


FIG. 4-10 - REPRESENTATION GRAPHIQUE DU NOMBRE COCYCLOMATIQUE NORMALISE EN FONCTION DU VOCABULAIRE

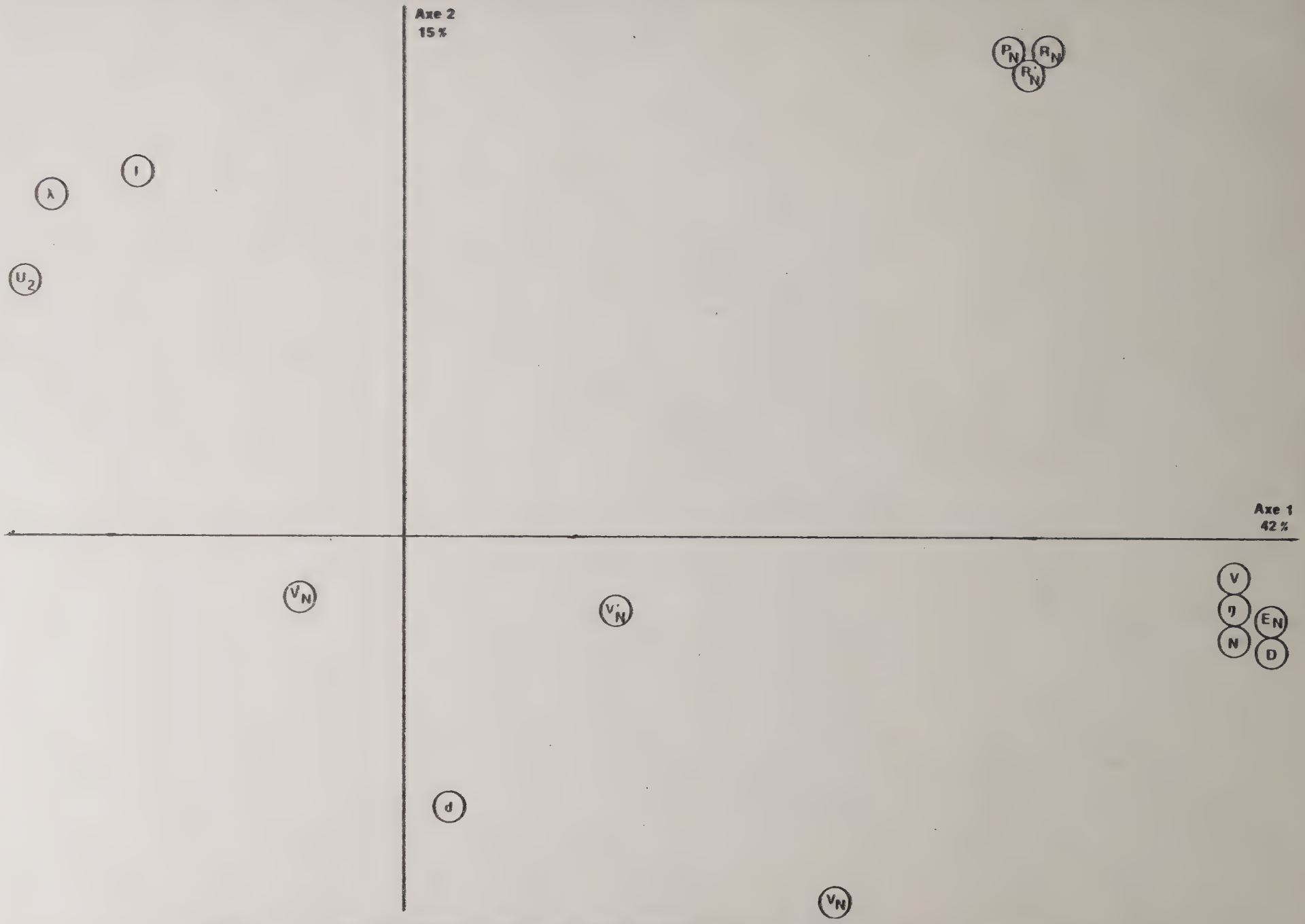


FIG. 4-11 - PROJECTION DES MESURES DERIVEES SUR LE 1er PLAN FACTORIEL

Mesures / Unités Fonctionnelles	U2	N	n	$V_{o\ell}$	λ	E_N	D	ν_N	V'_N	V_N	R'_N	R_N	P_N	ℓ	d
A	0.537	155.5	40.08	945.3	2.94	122.74	21.24	0.072	0.171	0.093	34.328	24.676	7.566	0.463	0.212
B	0.675	55.7	18.51	285.9	3.95	44.14	9.03	0.077	0.163	0.035	9.606	9.45	9.45	0.756	0.028
C	0.629	94.36	30.45	547.6	4.42	70.24	12.77	0.073	0.148	0.057	19.86	18.34	17.65	0.600	0.137
D	0.637	98.44	30.17	578.8	5.06	67.79	12.19	0.070	0.126	0.062	22.03	20.20	17.99	0.449	0.193
E	0.438	191.79	34.13	1115.8	2.56	146.04	25.94	0.064	0.151	0.081	3.36	2.16	1.21	0.415	0.168
F	0.466	126.13	31.69	693.8	2.60	94.02	17.82	0.05	0.126	0.069	35.97	34.73	33	0.442	0.174
G	0.509	189.05	44.08	1150	3.97	128.98	21.91	0.073	0.159	0.105	182.65	150.07	122.95	0.325	0.177
H	0.506	322.88	48.08	2074.45	3.85	163.34	26.73	0.051	0.182	0.063	50.66	18.94	7.86	0.594	0.252
K	0.441	207.6	48.95	1247.94	3.34	127.21	21.75	0.042	0.156	0.060	12.19	8.68	3.31	0.596	0.099
L	0.52	122.8	32.92	695.45	4.05	89	16.22	0.054	0.139	0.052	102.77	96.09	58.63	0.638	0.225
Z	0.621	82.39	28.17	444.42	3.00	71.70	14.05	0.087	0.174	0.109	0.57	0.402	0.150	0.366	0.273

FIG. 4-12 - VALEURS MOYENNES DES MESURES DE COMPLEXITE PAR UNITE FONCTIONNELLE

FIG. 4-13 - PROJECTION DES UNITES FONCTIONNELLES SUR LE 1er PLAN FACTORIEL

FIG. 4-14 - REPRESENTATION GRAPHIQUE DE L'ESTIMATEUR DE TAILLE PROPOSE PAR HALSTEAD EN FONCTION DE LA TAILLE

LTR

	m	σ	min	max	I de confiance
n_1	12.32	9.22	0	95	[11.68 ; 12.94]
N_1	70.83	115.66	0	1106	[62.93 ; 78.71]
n_2	20.02	20.37	0	156	[18.63 ; 21.41]
N_2	55.70	87.91	0	805	[49.70 ; 61.70]
n_u	7.03	13.16	1	137	[6.13 ; 7.93]
V	11.41	21.46	0	209	[9.94 ; 12.87]

MENTOR

	m	σ	min	max	I de confiance
n_1	12.23	8.24	2	81	[11.70 ; 12.76]
N_1	47.86	78.48	2	876	[42.83 ; 52.88]
n_2	15.14	18.56	1	166	[13.95 ; 16.33]
N_2	34.92	54.67	1	588	[31.42 ; 38.42]
n_u	5.77	8.80	1	98	[5.21 ; 6.33]
V	13.48	22.27	1	254	[12.05 ; 14.90]

FIG. 4.-15 - STATISTIQUES DE BASE DU COMPILATEUR LTR ET DU SYSTEME MENTOR

(LTR)

	n_1	N_1	n_2	N_2	n_u	V
n_1	1.00					
N_1	0.74	1.00				
n_2	0.78	0.87	1.00			
N_2	0.69	0.98	0.89	1.00		
n_u	0.76	0.88	0.74	0.82	1.00	
V	0.78	0.93	0.83	0.89	0.96	1.00

(MENTOR)

	n_1	N_1	n_2	N_2	n_u	V
n_1	1.00					
N_1	0.79	1.00				
n_2	0.70	0.86	1.00			
N_2	0.78	0.97	0.88	1.00		
n_u	0.77	0.91	0.87	0.89	1.00	
V	0.80	0.95	0.85	0.91	0.97	1.00

FIG. 4-16 - MATRICES DE CORRELATIONS

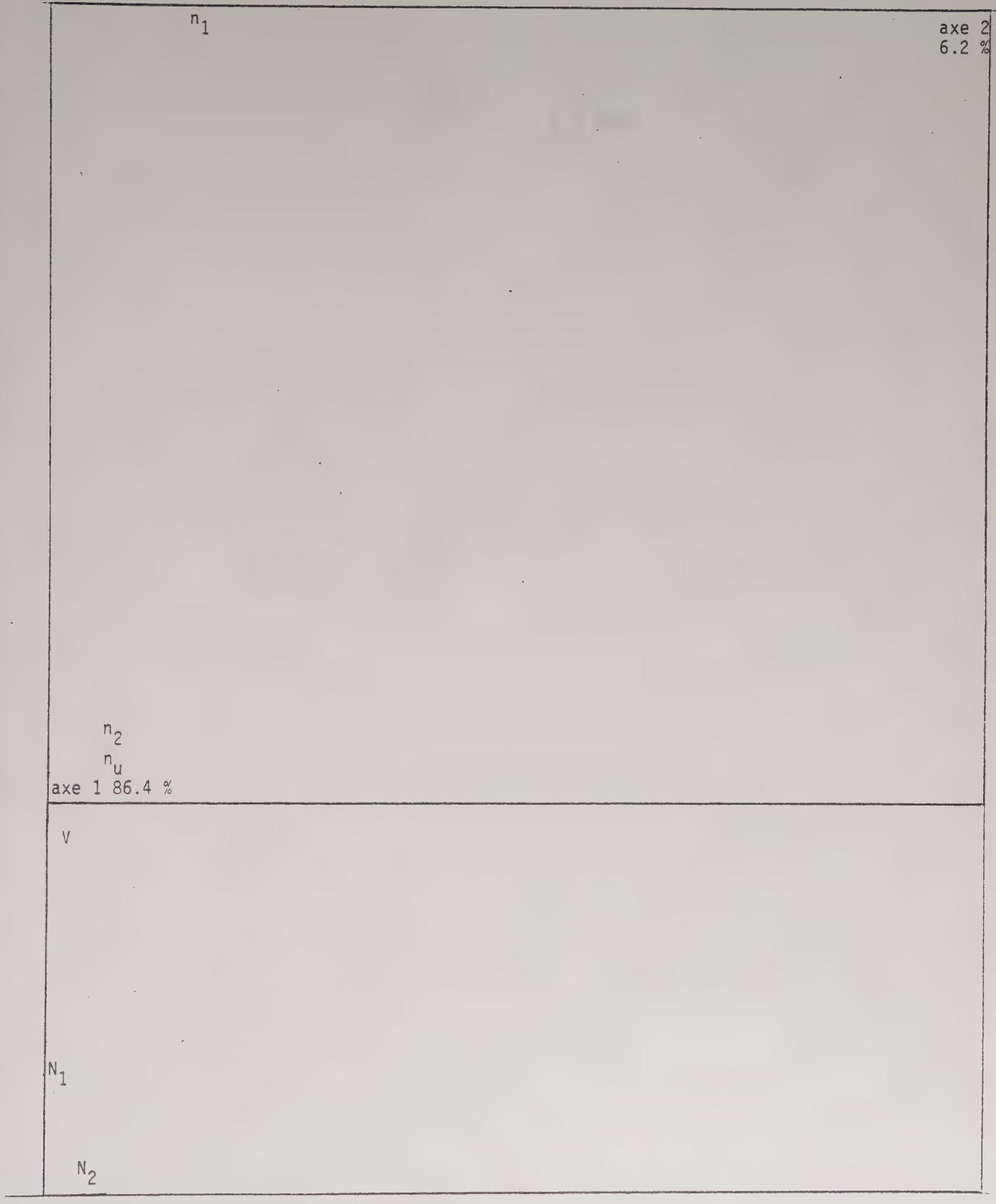


FIG. 4-17 - PROJECTIONS DES MESURES DE COMPLEXITE SUR LE

1er PLAN FACTORIEL (LTR)

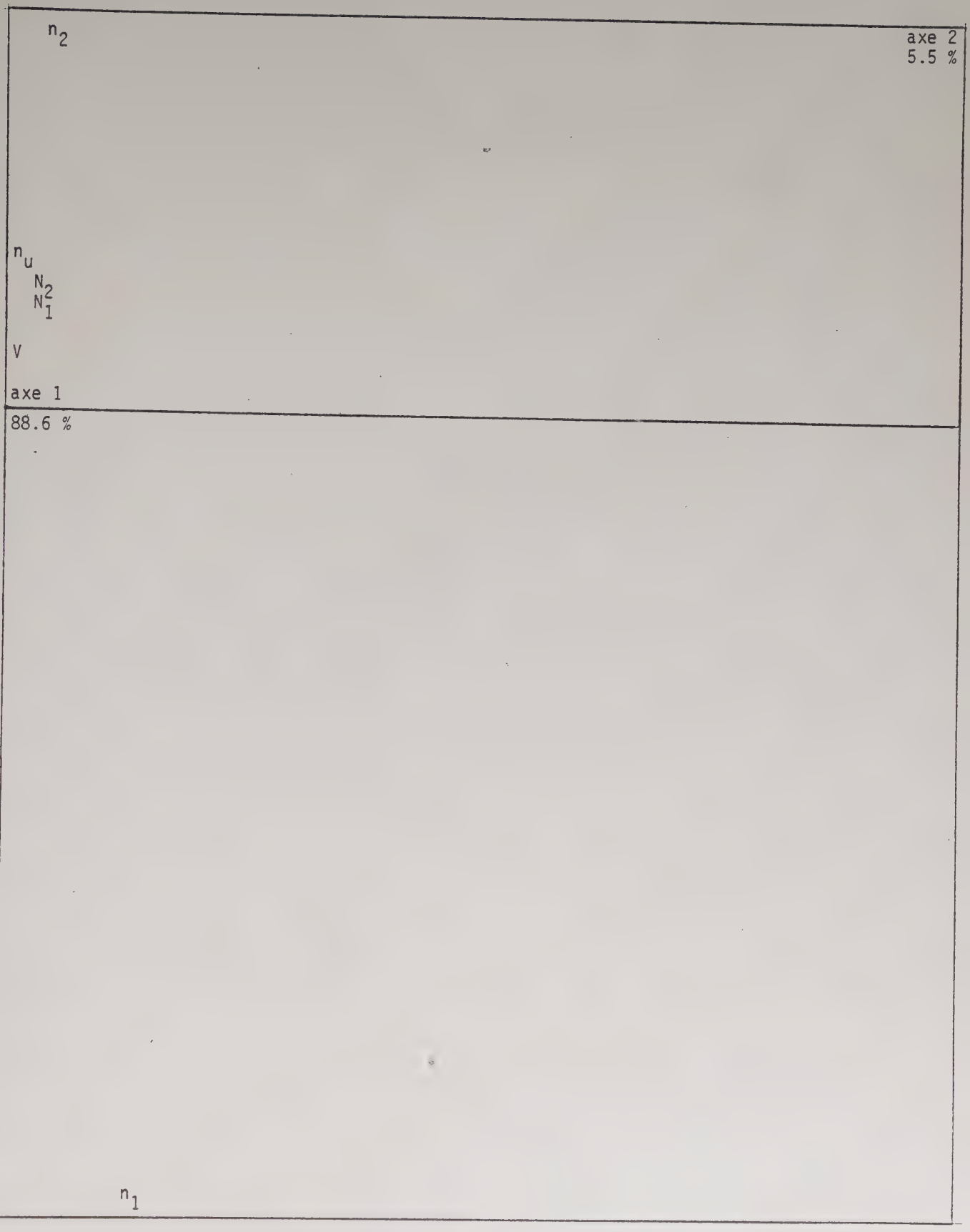


FIG. 4-18 - PROJECTION DES MESURES DE COMPLEXITE SUR LE 1er AXE FACTORIEL (MENTOR)

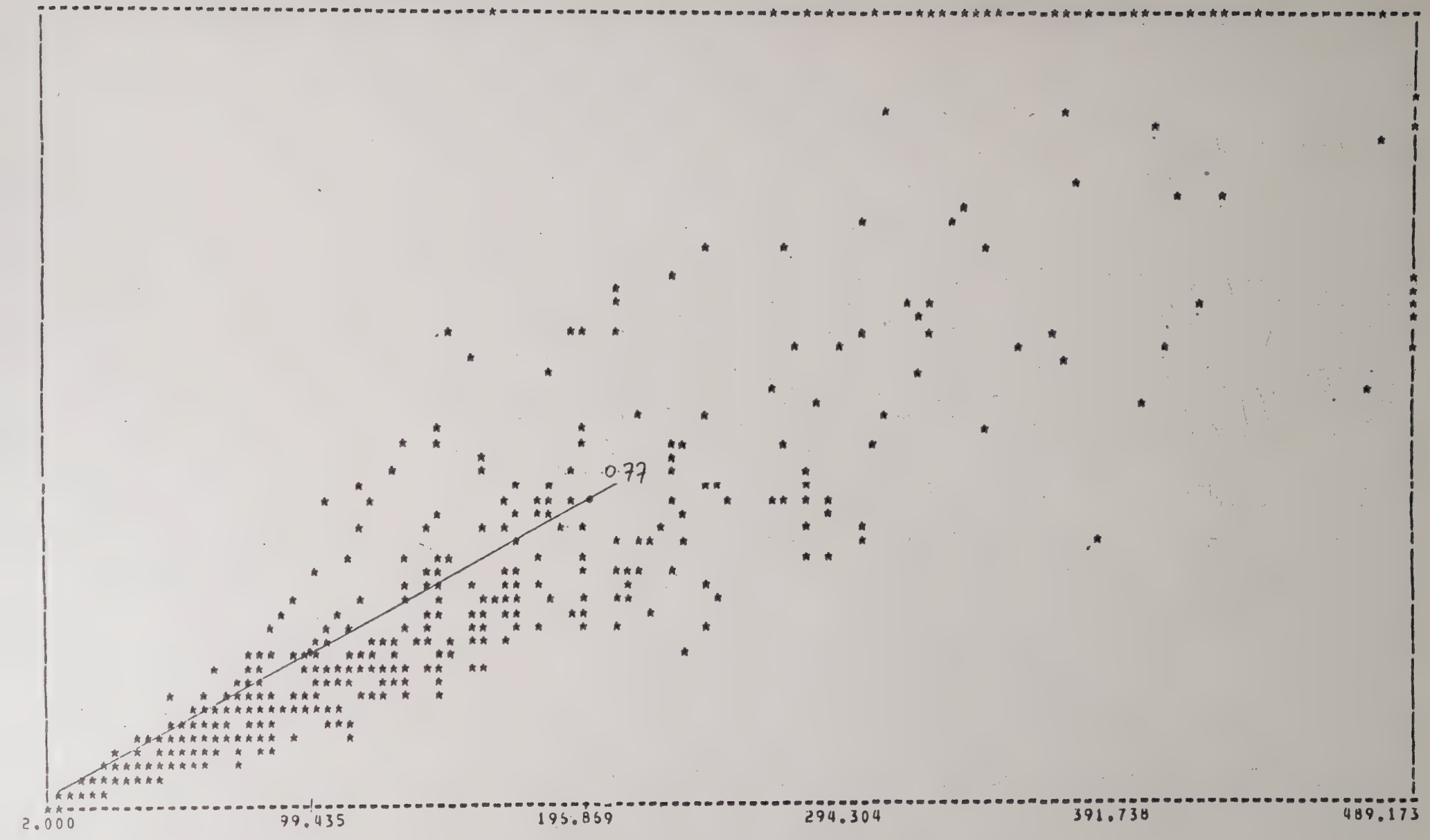


FIG. 4-19 - REPRESENTATION GRAPHIQUE DE L'ESTIMATEUR DE TAILLE

PROPOSE PAR HALSTEAD EN FONCTION DE LA TAILLE (MENTOR)

CHAPITRE 5

1 RELATIONS ENTRE LA COMPLEXITE ET LA FIABILITE

1.1 Existence directe de relations

1.1.1 position du problème

1.1.2 discrimination par valeurs moyennes

1.2 Caractérisation des relations complexité-fiabilité

1.2.1 discrimination non paramétrique

1.2.2 résultats quantitatifs et qualitatifs

1.3 Remarques

CHAPITRE 5

L'objet de ce chapitre est l'étude directe de relations entre la complexité d'un logiciel et sa fiabilité. D'un point de vue pratique le fait que des mesures effectuées sur le texte source d'un ensemble de procédures permettent de discriminer de façon significative entre les procédures possédant des erreurs et celles où de telles erreurs n'ont pas été détectées est important et apporte de nouveaux éléments pour la validation du logiciel.

Le premier paragraphe de ce chapitre sera consacré à la vérification directe de l'existence de relations entre données d'erreurs et mesures de complexité et le second à la caractérisation de ces relations.

1 RELATIONS ENTRE LA COMPLEXITE ET LA FIABILITE

1.1 Existence directe de relations

1.1.1 position du problème

Il importe de préciser en premier lieu les ensembles de données entre lesquels nous essayons d'établir des relations significatives.

Les erreurs ont été détectées par l'analyse des anomalies du logiciel étudié. Ces erreurs sont en partie caractérisées par leur origine, indiquant au cours de quelle phase de développement du logiciel elles ont été créées. Les données d'erreurs concernent ainsi des événements survenus au cours des différentes phases de production du programme, et, pour la plupart d'entre eux, avant que le programme atteigne sa forme finale.

En revanche, les mesures de complexité ont été obtenues en analysant le programme dans sa forme finale. Sous cette forme, le programme reflête de façon plus ou moins directe le contenu et les résultats de ses étapes successives de développement, selon que ces étapes sont plus ou moins proches du produit final. On peut donc s'attendre à ce que les erreurs créées au cours d'une phase de développement donnée soient d'autant mieux corrélées aux mesures de complexité que cette phase est proche du codage.

Afin de déterminer comment les mesures de complexité discriminent entre les procédures avec erreurs et les procédures sans erreur, suivant l'origine de ces erreurs, l'ensemble des procédures a été partitionné en trois paires de sous-ensembles:

- l'ensemble des procédures avec erreur(s),
  l'ensemble des procédures sans erreur;

- l'ensemble des procédures avec erreur(s) de codage ou de spécifications organiques,
  l'ensemble des procédures sans erreur de spécification organique ni erreur de codage;

- l'ensemble des procédures avec erreur(s) de codage,
  l'ensemble des procédures sans erreur de codage.

1.1.2 discrimination par valeurs moyennes

Les valeurs moyennes des mesures de complexité pour chaque sous-ensemble et pour l'ensemble de toutes les procédures ont été calculées dans le but d'indiquer le pouvoir discriminant de chaque mesure de complexité. Les résultats de ces calculs sont rassemblés sur la figure 5.1.

Une première observation peut être faite de l'analyse de la figure 5.1: toutes les mesures de complexité s'avèrent nettement discriminantes, sauf le nombre cyclomatique et le nombre cocyclomatique normalisés. La valeur du nombre cyclomatique des procédures avec erreurs et des procédures sans erreur est dans un rapport excédant légèrement trois. Cette valeur a été également obtenue par Schneidewind <ScHo79>. La suggestion de Mac Cabe d'écrire des procédures telles que le nombre cyclomatique correspondant n'excède pas dix semble être ainsi largement confirmée sur les données du compilateur LTR puisque le nombre cyclomatique des procédures sans erreur vaut en moyenne moins de quatre et celui des procédures avec erreurs plus de treize.

L'absence de pouvoir discriminant du nombre cyclomatique normalisé prouve que ce pouvoir discriminant est surtout dû à sa forte composante de taille. La complexité structurelle des procédures est davantage expliquée par l'atteignabilité ou le nombre de chemins, qui, même une fois normalisés par la taille des procédures, discriminent de façon significative les procédures avec erreurs des procédures sans erreur.

La normalisation de ces deux mesures de complexité montre, sur l'échantillon de données du compilateur LTR, qu'elles sont complémentaires du nombre cyclomatique lors de l'évaluation statique de la complexité d'un programme. Cette remarque permet de modifier la conclusion de l'étude de Schneidewind (cf: CHAPITRE 1).

Une seconde observation est que le pouvoir discriminant des mesures de complexité est maximum lorsque sont considérées les erreurs créées lors du codage ou imputables à des spécifications organiques incorrectes. Ce résultat rappelle simplement que ces deux étapes déterminent de façon directe l'organisation et la structure du programme final.

D'autre part, la discrimination est moins significative lorsqu'elle s'applique uniquement aux erreurs de codage. Ce résultat est cohérent avec le fait que si toutes les spécifications ont été menées correctement, le codage devient un travail de transcription de ces spécifications. Les erreurs introduites sont surtout alors des erreurs "matérielles", moins directement liées à la structure de la procédure à coder. Les conclusions déduites de l'analyse des figures 5.1 confirment donc l'existence de relations nettes entre la complexité d'un ensemble de procédures et sa fiabilité.

1.2 Caractérisation des relations complexité-fiabilité

1.2.1 discrimination non paramétrique

Afin d'affiner ces premières conclusions, il est nécessaire de quantifier plus rigoureusement les discriminations réalisées par les mesures de complexité, et de classer ces mesures suivant leur pouvoir discriminant.

> - Lorsque par exemple la taille d'une procédure est petite et que son nombre de chemins normalisé est grand, doit-on considérer la procédure comme fiable ou comme suspecte ?.

La méthode de discrimination non paramétrique, <CeLe80>, hiérarchise la pertinence des mesures de complexité suivant le critère étudié. Cette méthode a été utilisée sur les trois couples de sous-ensembles de procédures, chaque procédure étant caractérisée par ses mesures de complexité. Pour chaque couple de sous-ensembles, le résultat de l'analyse discriminante se présente sous la forme d'un arbre de décision, chaque noeud de l'arbre correspondant à une des mesures de complexité, et les noeuds terminaux décrivent la répartition des procédures de l'échantillon suivant leurs caractéristiques de fiabilité.

1.2.2 résultats quantitatifs et qualitatifs

Les figures 5.2 donnent les trois arbres de décision relatifs aux trois couples de sous-ensembles de procédures définis précédemment. Chaque arbre fournit un algorithme pour la validation d'une procédure nouvellement produite: étant données les mesures de complexité de cette procédure, le parcours de l'arbre conduit à un noeud terminal et la situation de ce noeud terminal permet d'estimer la qualité de la procédure étudiée.

Les algorithmes de discrimination entre les procédures avec erreurs et les procédures sans erreur rendent compte du plus grand pouvoir discriminant des mesures textuelles. Les algorithmes de discrimination montrent que la complexité textuelle identifie l'alternative présence ou absence d'erreurs. L'alternative plus fine: présence ou absence de types d'erreurs particuliers (à travers les phases d'origine correspondantes), nécessite une caractérisation plus précise et la prise en compte de la complexité structurelle.

La figure 5.3 indique l'évolution du type de complexité discriminant en fonction de la finesse de l'alternative. Le rapport hiérarchique des types de complexité établi au chapitre précédent apparait de nouveau: la complexité structurelle précise une alternative lorsque la complexité textuelle en assure le bien fondé.

Une deuxième observation de la figure 5.2 est le résultat du vocabulaire, mesure de complexité la plus discriminante, alors qu'Halstead préconisait plutôt le volume, comme conjonction des mesures taille et vocabulaire. En fait le coefficient de 0.99 entre le volume et la taille prouve que ces deux mesures sont proportionnelles; l'application de la fonction log(2) réduit la plage de valeurs du vocabulaire jusqu'à devenir une constante, représentée par la pente de la droite de la figure 5.4. La notion de vocabulaire n'existe plus dans la définition du volume et la

variable taille est moins discriminante que le vocabulaire (on peut se convaincre de ce dernier point en constatant que la taille des procédures du compilateur LTR a été correctement estimée à l'aide du nombre d'opérateurs distincts et d'opérandes distinctes de chaque procédure, soit de son vocabulaire).

Enfin, le seuil de discrimination au premier noeud des arbres apporte une information qu'il peut être intéressant d'interpréter. La figure 5.2 indique que le vocabulaire est la variable la plus discriminante avec un seuil de discrimination égal à 39. La validation du prédicteur de taille connaissant le vocabulaire permet de montrer que ce seuil de 39 correspond à environ une cinquantaine de lignes de code source pour l'échantillon considéré. On retrouve ainsi expérimentalement la règle empirique suivant laquelle une procédure écrite en langage de haut niveau ne devrait pas excéder une page de listing.

1.3 Remarques

Le chapitre 3 a montré qu'à chaque type d'erreurs était associée une phase d'origine. La caractérisation de chaque phase d'origine par un profil de complexité permettrait de prévoir les types d'erreurs probables et d'orienter l'effort de test. Une discrimination paramétrique a prouvé que cette hypothèse était trop forte.

La mesure N1/N2, qui pourtant peut identifier un style de programmation, est une mesure non discriminante n'ayant pas de contribution significative au cours de l'analyse des mesures de complexité.

	η_1	N_1	η_2	N_2	ν	V'(*)	V(**)	P	R'	
FICHIER TOTAL	12,30	70,74	19,99	55,63	7,02	20,32	11,39	20507,23	29902,99	
$\overline{ANO}$	9,58	38,28	12,97	30,82	3,74	10,37	5,53	4465,75	6629,26	TABLEAU 1
ANO	17,36	131,16	33,06	101,83	13,12	38,83	22,31	50369,92	73229,17	
$\overline{PO}$	10,24	46,57	14,78	37,60	4,50	12,50	6,85	7268,83	10133,58	
PO	18,34	141,98	35,30	108,77	14,44	43,37	24,80	59617,59	88311,43	TABLEAU 2
$\overline{P}$	11,20	56,89	17,21	45,56	5,42	15,79	8,62	12287,26	17333,86	
P	18,63	151,05	36,13	114,05	16,26	46,58	27,52	68366,81	103097,00	TABLEAU 3

(*) NOMBRE COCYCLOMATIQUE ETENDU

(**) NOMBRE COCYCLOMATIQUE

FIG. 5-1 - VALEURS MOYENNES DES DIFFERENTES MESURES DE COMPLEXITE SUR LES GROUPES DE PROCEDURES (PAGE 1/3)

	R	TC	U_2	N	η	V(*)	λ	E_N	D	ν_N	
FICHIER TOTAL	25750,91	63,12	0,5652	131,27	33,53	771,99	3,91	92,14	16,45	0,06588	
$\overline{ANO}$	6155,97	34,79	0,6087	72,30	23,59	374,11	4,32	56,72	11,35	0,06998	TABLEAU 1
ANO	62228,69	115,85	0,4860	238,77	51,67	1497,22	3,15	156,71	25,74	0,06125	
$\overline{PO}$	9236,72	42,78	0,5921	93,00	27,43	513,49	4,13	70,62	13,33	0,06804	TABLEAU 2
PO	74538,18	123,06	0,4303	403,27	69,69	2676,22	2,69	238,12	36,84	0,06407	
$\overline{P}$	15369,2	50,57	0,5472	168,77	38,74	1030,91	3,70	114,87	19,63	0,06547	TABLEAU 3
P	86198,01	135,93	0,4775	296,13	58,97	1920,59	3,08	186,90	29,68	0,07054	

(*) VOLUME

FIG. 5-1 - (PAGE 2/3)

	V'_N (*)	V_N (**)	R'_N	R_N	P_N	1	t	
FICHIER TOTAL	0,14968	0,06816	52,99	44,89	34,67	0,522	0,161	
$\overline{\text{ANO}}$	0,14249	0,05851	15,41	14,09	9,88	0,553	0,121	TABLEAU 1
ANO	0,16278	0,08577	121,48	101,02	79,85	0,465	0,233	
$\overline{\text{PO}}$	0,14415	0,06090	20,75	18,28	13,23	0,546	0,153	TABLEAU 2
PO	0,17686	0,09659	231,48	182,44	138,83	0,447	0,166	
$\overline{\text{P}}$	0,14948	0,06936	78,77	69,61	49,69	0,515	0,172	TABLEAU 3
P	0,18042	0,09767	166,33	120,81	88,15	0,456	0,147	

(*) NOMBRE COCYCLOMATIQUE ETENDU/TAILLE

(**) NOMBRE COCYCLOMATIQUE/TAILLE

FIG. 5-1 - (PAGE 3/3)

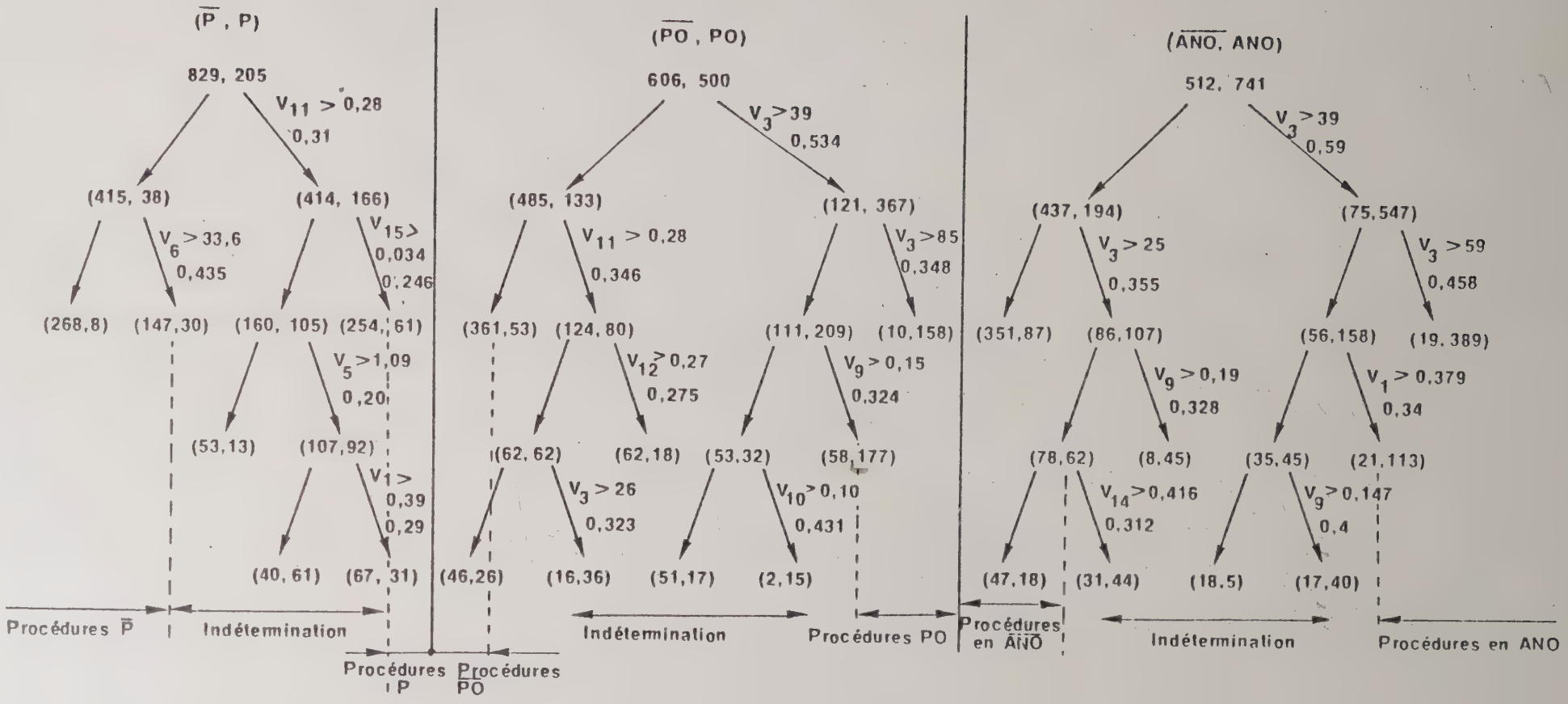


FIG. 5-2 - ARBRES DE DECISIONS SUR LA PRESENCE D'ANOMALIES

TYPES / COMPLEXITES	1		2		3	
	$\overline{\text{ANO}}$	ANO	$\overline{\text{PO}}$	PO	$\overline{\text{P}}$	P
Textuelle	Oui		Oui		Non	
Structurelle	Non		Oui		Oui	

FIG. 5-3 - EVOLUTION DU TYPE DE COMPLEXITE EN FONCTION DES DÉCISIONS

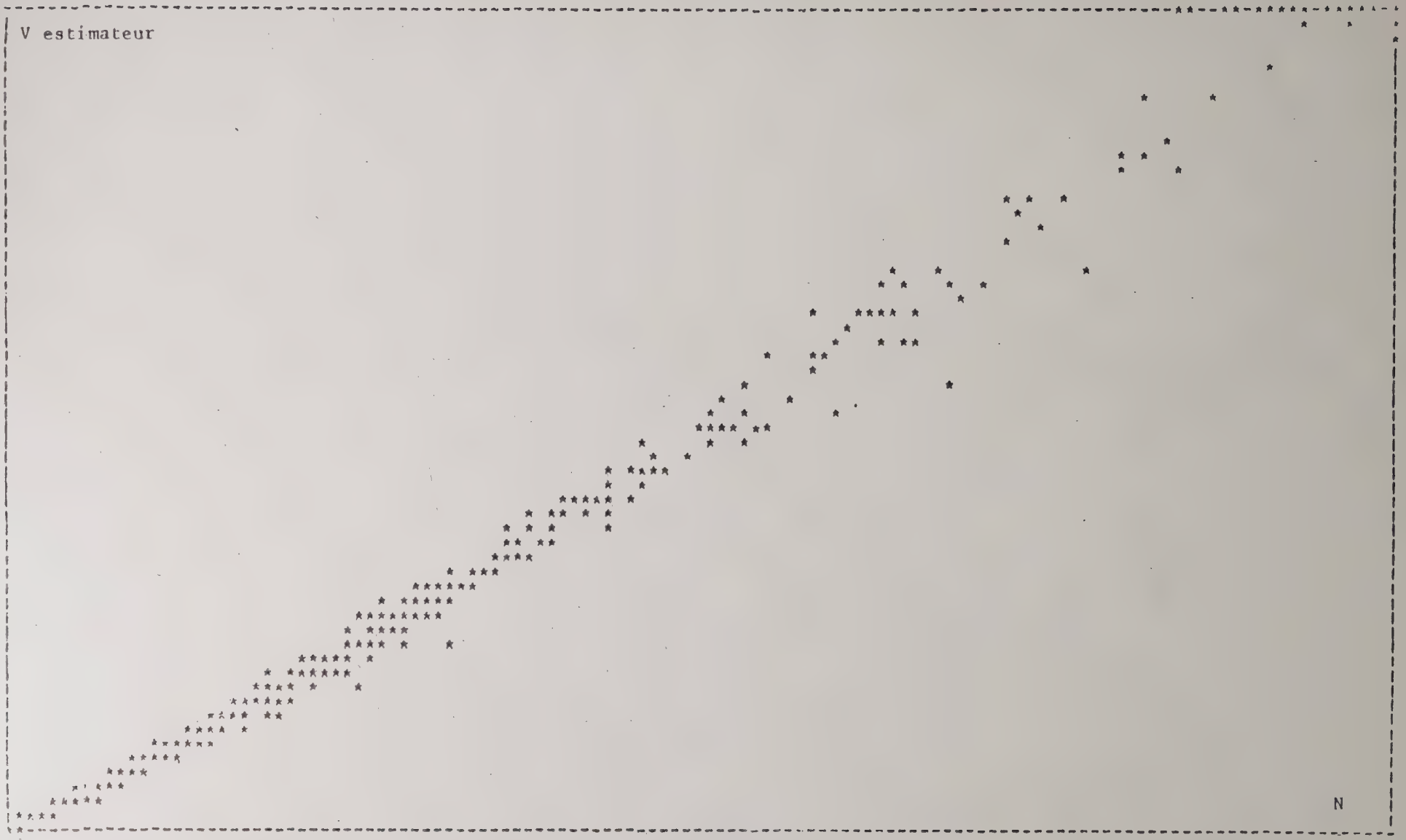


FIG. 5-4 - REPRESENTATION GRAPHIQUE DU VOLUME EN FONCTION DE LA TAILLE

CHAPITRE 6

1 RESULTATS DE L'ETUDE ET CONCLUSIONS

CHAPITRE 6

1 RESULTATS DE L'ETUDE ET CONCLUSIONS

Les analyseurs de programmes relevant les mesures textuelles et structurelles de programmes sont maintenant disponibles dans les principaux langages. Ils permettent de prélever les mesures d'un programme et de détecter plus tôt les procédures probablement correctes ou pathologiques suivant le respect ou non des normes de programmation établies. Ils permettent enfin de donner une mesure quantitative de la qualité d'un logiciel en termes de complexité.

Le principal résultat de l'étude a été de dégager des liaisons quantitatives et qualitatives entre la complexité d'une procédure et sa fiabilité.

Les conclusions obtenues au cours des traitements statistiques sont relatives à un logiciel, elles doivent pour la plupart être confirmées sur d'autres programmes.

Le profit que la CIMSA peut dégager de cette étude est intéressant à noter. La CIMSA possède sur un de ses logiciels des critères quantitatifs entre les deux attributs complexité et fiabilité. Il est légitime qu'elle cherche à les "transporter" sur d'autres produits. On pourrait penser que le "transport" des résultats quantitatifs nécessite de reconduire les conditions de développement du compilateur LTR sur de futurs logiciels, mais ceci ne tient pas compte de la réalité logicielle. Soumises à de continuelles évolutions et à de fréquentes améliorations, il serait à mon avis négatif de figer les conditions de développement de tout produit logiciel sous le prétexte que des relations quantitatives seraient mieux conservées. Pour la CIMSA, la suite naturelle de cette étude est bien plutôt de l'adapter à de nouveaux projets. La principale règle méthodologique à observer, pour une application raisonnable de cette étude, est d'apprécier l'ampleur et l'influence des modifications de conditions de développement sur les relations qualitatives et quantitatives établies. Une démarche en ce sens est déjà entreprise à la CIMSA sur un logiciel LTR existant dont on possède les relevés d'erreurs. Les relations quantitatives obtenues lors de l'analyse du compilateur LTR sont testées, par exemple en comparant les discriminations réelles entre procédures avec erreur(s) et procédures sans erreur avec les discriminations calculées à l'aide des arbres discriminants de la figure 5.2.

Pour toute autre compagnie soucieuse de faire des études de fiabilité du logiciel, l'utilisation de cette étude est différente. Les conditions de développement d'un produit logiciel CIMSA ne sont pas traduisibles ni sérieusement importables. En conséquence, le dégagement de concepts méthodologiques adéquats, intervenant dans l'historique complet d'une étude de fiabilité, semble être l'utilisation pratique de cette étude. Sa portée devient évidemment plus limitée.

Trois remarques apparaissent au terme de l'étude:

- La plus importante est la mise en évidence expérimentale qu'une procédure du compilateur LTR excédant une page de listing contenait probablement des erreurs. Les problèmes d'interface inhérents au découpage d'un programme en plusieurs modules n'ayant pas été traités il resterait à vérifier que le découpage représente bien un progrès et non un déplacement des difficultés de la programmation.
- L'existence de deux classes d'unités fonctionnelles, celles de la souche et celles du générateur, leur fonctionnalité et leur historique de développement ont été établis au cours de l'analyse statistique des données d'erreurs et des mesures de complexité.
- Un des bénéfices de cette étude aura été l'écriture d'une procédure formelle, entièrement automatisée, d'enregistrement d'erreurs au moment de leur correction: l'identification et le codage des erreurs dès leur traitement améliorent doublement la qualité des données d'erreurs.

Certaines conclusions publiées dans la littérature corroborent les résultats trouvés au cours de l'étude ou constituent une extension naturelle de ceux-ci.

1) Le rôle de référence joué par les procédures sans erreur oblige à éliminer celles qui semblent telles du fait de l'absence de test (on peut évaluer l'effort de test par le pourcentage de chemins testés). Motley montre que l'effort de test en fonction du nombre de lignes de code décroit au delà d'une certaine valeur <MoBr77>; ce qui milite à nouveau indirectement en faveur de procédures de courtes tailles.

2) La définition d'un ensemble de mesures de complexité de base et l'identification des caractéristiques des programmes et des langages afin d'entreprendre des études comparatives constituent aussi deux propositions fréquemment évoquées et qui paraissent opportunes au terme de cette étude.

3) Enfin, les deux propositions énoncées dans les ouvrages de Thayer ou Motley peuvent être largement reprises:

- nécessité de définir une nouvelle classification d'erreurs moins orientées sur le codage avec des codes complémentaires d'erreurs de spécifications afin de couvrir toute la durée de développement d'un produit logiciel, semblable à celle proposée à la figure 6.1,
- nécessité de mettre au point dès les phases de spécification les techniques actuellement utilisées afin de prélever des mesures au cours des différentes phases du cycle de vie pour mieux contrôler la complexité et être capable de "faire remonter" les études de fiabilité du niveau Codage au niveau Spécifications. Ce suivi pas à pas des programmes représente un axe d'études immédiates très prommetteur, depuis longtemps envisagé mais jamais réellement abordé.

En résumé, l'évaluation du logiciel nécessite des outils de mesures, des modèles, des données. Même s'il n'existe pas d'ensemble de

mesures de base à appliquer à toute étude de fiabilité, nous savons que des outils automatiques de mesures de programmes écrits dans les principaux langages sont disponibles. De nombreux modèles de fiabilité sont proposés; même s'ils évaluent la fiabilité d'un logiciel à postériori plutôt qu'à priori, ces modèles ont davantage besoin d'être confrontés à des données expérimentales que d'être étendus. Les données expérimentales utilisées doivent être simples, complètes, faciles à recueillir et universelles afin qu'avec les caractéristiques des langages les uns par rapport aux autres, des liaisons entre les facteurs de l'environnement de programmation et la fiabilité soient dégagées. Dans le cadre d'un projet bien spécifié, ces données d'erreurs représentent "un sous produit de la programmation" et n'élèvent pas considérablement le coût du produit.

Les différentes approches d'évaluation de la fiabilité du logiciel ne s'excluent pas et peuvent être utilisées concurremment au cours des différentes phases de développement d'un produit logiciel. Le schéma suivant, immédiatement appliquable, pourrait être proposé <MeRa79>:

- mesure de la complexité des modules lors de leur écriture (détermination des caractéristiques de la Science du Logiciel); le résultat est le rejet si la complexité dépasse un seuil admissible donné.
- au cours de la phase de test évaluation et suivi de l'évolution de la fiabilité à l'aide d'un des modèles de la théorie de la fiabilité du matériel.
- à la fin de la phase de test évaluation de l'efficacité des jeux de test utilisés au moyen de la théorie de la complexité de la structure vis à vis du comportement à l'exécution (détermination du pourcentage de chemins testés par exemple).

K - CAHIER DES CHARGES

K000	Cas non prévu (code à éviter)
K100	Fonction inutile
K200	Fonction manquante
K300	Fonction (ou groupe de fonctions) non conforme au besoin de l'utilisateur
K400	Fonction à redéfinir

L - SPECIFICATIONS FONCTIONNELLES

L000	Cas non prévu (code à éviter)
L100	Interface incorrectement défini entre deux fonctions (procédures, sous programme, etc...)
L200	Oubli d'un traitement dans une fonction (Ex. : adaptation à une nouvelle machine)
L300	Fonction manquante
L400	Outil manquant

M - SPECIFICATION ORGANIQUE

M000	Cas non prévu (code à éviter)
M100	Erreur dans un élément logique d'interface (Ex. : initialisation de tables...)
M200	Erreur dans un élément physique d'interface (Ex. : mauvais chaînage pour transmissions, structure de données incompatibles avec certains périphériques...)
M300	Cas particuliers oubliés (ex. : cas d'erreurs)
M400	Insuffisance ou imprécision d'analyse donnant des erreurs d'interprétation lors du codage
M500	Mauvaise structuration des données (Ex. : format de tableaux difficilement – ou pas du tout – compatible avec le langage utilisé)
M600	Séquence de traitement à reprendre entièrement (ex. : intégration de corrections précédentes)
M610	Séquence de traitement inefficace ou inutile
M700	Erreur d'analyse d'une séquence de tests

N - MISE EN OEUVRE

N000	Cas non prévu (code à éviter)
N100	Incompatibilité de fonctionnement avec un processeur existant
N200	Inadéquation entre le produit et les accès aux ressources physiques de la machine
N300	Erreur ou absence de commandes dans le flot d'entrée
N400	Module mal placé (ex. : pour obtenir un logiciel correspondant à la configuration physique)
N500	Mauvaise structuration de programme à intégrer.

FIG. 6-1 - CODES COMPLEMENTAIRES D'ANOMALIES

ANNEXE

1 TECHNIQUES DESCRIPTIVES

1.1 Test d'indépendance

1.2 Calculs préliminaires

1.2.1 analyse dans un espace

1.2.2 analyse dans l'autre espace

1.3 Analyse en Composantes Principales

1.4 Analyse Factorielle des Correspondances

2 TECHNIQUES EXPLICATIVES

2.1 Régression multiple

2.1.2 modèle général

2.2.2 modèle gaussien

2.2 Analyse discriminante

2.2.1 discrimination paramétrique

2.2.2 discrimination non paramétrique

ANNEXE

Ce chapitre présente toutes les méthodes statistiques utilisées dans cette étude: le test du khi-2 <LeFe75>, l'analyse en composantes principales <LeFe75, LeMoTa, Sch78>, l'analyse factorielle des correspondances <LeFe75, LeMoTa, Sch78>, le modèle de régression multiple <DuIn79>, l'analyse discriminante <CeLe80>. Ces méthodes appartiennent à deux familles d'outils statistiques:

- les outils descriptifs dont l'analyse en composantes principales et l'analyse factorielle des correspondances sont deux exemples,
- les outils explicatifs avec le modèle de régression et l'analyse discriminante.

Il est possible de ranger le test du khi-2 parmi les outils explicatifs puisqu'il détermine si deux partitions d'un même ensemble sont indépendantes. Toutefois il semble plus cohérent d'associer le test du khi-2 aux techniques descriptives car cet outil analyse le même tableau de données et utilise la même distance que ceux employés par l'analyse factorielle des correspondances.

Ce chapitre s'articule en deux paragraphes. Le premier présente les techniques descriptives et le second les techniques explicatives.

1 TECHNIQUES DESCRIPTIVES

Un des attraits de ces techniques est leur absence d'hypothèses et en conséquence leur facilité d'emploi. La description d'un tableau de données est une phase initiale et une étape obligatoire dans toute étude statistique. Nous verrons que les méthodes présentées permettent des analyses de données poussées.

Soit un tableau croisé de données réelles t(i,j). Soient i et j deux indices, le premier variant de 1 à n et le second de 1 à p. Notons:

$$t(i,.) = \Sigma\, t(i,j)$$
$$t(.,j) = \Sigma\, t(i,j)$$
$$t(.,.) = \Sigma\, t(i,j)$$
$$p(i,j) = t(i,j)/t(.,.)$$
$$p(i,.) = t(i,.)/t(.,.) = \Sigma\ p(i,j)$$
$$p(.,j) = t(.,j)/t(.,.) = \Sigma\ p(i,j)$$

Nous déduisons que $p(.,.) = 1$

1.1 Test d'indépendance

Les tableaux de contingence résument la correspondance existant entre deux partitions d'un ensemble de t(.,.) individus. Le test du khi-2 effectué sur le tableau de contingence permet de déterminer si les partitions sont indépendantes. Le test du khi-2 compare la probabilité

empirique p(i,j) à la probabilité théorique, en cas d'indépendance p(i,.)*p(.,j), de la façon suivante:

$$T = \sum_{ij} \langle (t(i,j) - t(.,.)*p(i,.)*p(.,j))**2 \rangle / (t(.,.)*p(i,.)*p(.,j))$$
$$T = t(.,.) * \sum_{i,j} \langle (p(i,j) - p(i,.)*p(.,j))**2 \rangle / (p(i,.)*p(.,j))$$

On démontre que T suit en cas d'indépendance une loi de Khi-2 à (n-1)*(p-1) degrés de liberté. A un niveau d'acceptation alpha donné on lit dans les tables du Khi-2 la valeur de Khi-2<(n-1)*(p-1)> et l'on obtient la règle de décision suivante:

- Si T > Khi-2<(n-1)*(p-1)> les deux partitions sont dépendantes,
- Si T < Khi-2<(n-1)*(p-1)> les deux partitions sont indépendantes .

Nous présentons deux méthodes descriptives d'analyse multidimensionnelle:

- l'analyse en composante principales,
- l'analyse factorielle des correspondances.

Il s'agit dans les deux cas de projeter le tableau de données sur un espace de très faible dimension un, deux, trois tel que la perte d'informations au cours de cette réduction d'espace soit minimale. Nous commençons la présentation de ces méthodes par un paragraphe de calculs préliminaires valables pour les deux techniques.

1.2 Calculs préliminaires

Supposons n>p et effectuons l'analyse dans l'espace Rp.

1.2.1 analyse préliminaire dans l'espace Rp

Le tableau T(n*p) représente alors un nuage de n points. Cherchons un espace de dimension un (droite) qui réalise le meilleur ajustement possible du nuage des n points. Il faut pour cela chercher la droite D de vecteur directeur u qui minimise les distances m(k)h(k), m(k) étant un point du nuage, h(k) sa projection sur la droite D, o l'origine des axes, k variant de 1 à n. On a les relations suivantes:

$$(m(k)h(k))**2 = (om(k))**2 - (oh(k))**2$$
$$\sum_k (m(k)h(k))**2 = \sum_k (om(k))**2 - \sum_k (oh(k))**2$$

De cette deuxième égalité, nous reconnaissons l'expression de l'inertie du nuage des n points, expression constante dans la recherche de la droite D. Minimiser $\Sigma(m(k)h(k))**2$ revient à maximiser $\Sigma(oh(k))**2$. oh(k) représente la projection du kième point sur la droite D; ce n'est rien d'autre que la kième composante du vecteur Tu, d'où minimiser $\Sigma(m(k)h(k))**2$ revient à maximiser u'T'Tu. La contrainte u unitaire n'enlève rien à la généralité du problème.

La droite D qui réalise le meilleur ajustement dans Rp du nuage des n points porte le vecteur directeur u qui doit maximiser l'expression:

$$(E1) = u'T'Tu - lambda1*(u'u-1)$$

Nous avons à maximiser par la méthode des multiplicateurs de Lagrange. Un simple calcul montre:

$$u'T'Tu = \sum_{jj'} u(j)*u(j')*< \sum_{k} t(k,j)*t(k,j')>$$

ce qui permet de déduire:

$$\partial(E1)/\partial u(1) = 2*(T'Tu)(1)$$

On déduit simplement le résultat suivant:

$$\partial(E1)/\partial u = 2*T'Tu - 2*lambda1*u$$

En multipliant cette dernière expression par u', on obtient u'T'Tu=lambda1 où u'T'Tu est précisément la quantité à maximiser. Nous déduisons:

- lambda1 est la plus grande valeur propre de T'T,
- u est le vecteur propre de T'T associé à lambda1.

Nous appellerons maintenant le vecteur u trouvé précédemment u1. Si nous cherchons le plan qui ajuste au mieux le nuage des n points, on montre par l'absurde que u1 appartient à ce plan. Sans limiter la généralité du problème, nous allons chercher le vecteur u, orthogonal à u1, unitaire tel que l'expression (E2) soit maximisée:

$$(E2) = u'T'Tu - lambda2*(u'u-1) - mu2*(u'u1)$$

De la même façon que précédemment, nous avons:

$$\partial(E2)/\partial u = 2*T'Tu - 2*lambda2*u - mu2*u1$$

En multipliant cette dernière expression par u1', on déduit que mu2 = 0 et les deux résultats suivants:

- lambda2 est la deuxième plus grande valeur propre de T'T,
- u est le vecteur propre de T'T associé à lambda2.

Par récurrence, on démontrerait le résultat:

- L'espace V de dimension q, q<p, réalisant le meilleur ajustement du nuage des n points est engendré par les q vecteurs propres de la matrice T'T associés aux q plus grandes valeurs propres lambda1> lambda2> ...> lambdaq.

Nous allons démontrer maintenant que l'analyse dans l'espace des individus est sous-jacente à l'analyse dans l'espace des variables.

1.2.2 analyse préliminaire dans l'espace Rn

La matrice T'(p*n) décrit le nuage des p variables dans l'espace des individus. Nous cherchons encore le vecteur v de Rn, unitaire, tel que l'expression (E3) soit maximum:

(E3) = v'TT'v - alpha1*(v'v-1)

On déduit:

alpha1 est la plus grande valeur propre de TT',
v est le vecteur propre associé à alpha1.

Montrons que le vecteur de Rp, T'v, est vecteur propre de T'T. Nous avons les deux relations:

T'Tu1 = lambda1*u1 (1)
TT'v1 = alpha1 *v1 (2)

En multipliant (1) par T et (2) par T' on démontre que alpha1 est égal à lambda1, et que le vecteur Tu1 est vecteur propre de TT' dans Rn (de même T'v1 est vecteur propre de T'T dans Rp) de norme lambda1. On obtient le résultat suivant par récurrence:

- Toutes les valeurs propres non nulles de deux matrices T'T et TT' sont égales (et avec le même ordre de multiplicité).
- Les q vecteurs propres de l'espace Rn (v1, ..., vq) se déduisent des q vecteurs propres de l'espace Rp (u1, ..., uq) par les formules:
  vk = 1/(lambdak**(1/2)) * Tuk pour k variant de 1 à q
- Les q vecteurs propres (u1, ..., uq) de l'espace Rp se déduisent des q vecteurs propres de l'espace Rn (v1, ..., vq) par les formules:
  uk = 1/(lambdak**(1/2)) * T'vk pour k variant de 1 à q

Il est clair maintenant que l'analyse statistique de tableaux de données s'effectue toujours dans l'espace de plus petite dimension.

1.3 Analyse en Composantes Principales (ACP)

L'ACP s'applique en général aux tableaux de mesures, l'analyse se faisant en rapportant au centre de gravité du nuage de points individus l'origine des axes.

Les variables du tableau de mesures peuvent être d'échelles très diverses, de natures différentes et l'on peut désirer leur accorder la même importance. Pour ces raisons on normalise le tableau T en le remplacant par X, de terme général x(i,j), de la façon suivante:

x(i,j) = (t(i,j)-t̂(.,j))/(n**(1/2)*s(j)) où:
t̂(.,j) = t(.,j)/n
s(j)**2 = $\sum_i$ (t(i,j)-t̂(.,j))**2/n

Nous devrons dans l'espace Rp diagonaliser la matrice C=X'X dont le terme général c(j,j') est exactement le facteur de corrélation entre les variables j et j'. Dans l'espace Rn, la distance de deux variables j et j' vaut:

$$d(j,j')=2*(1-c(j,j')).$$

Ainsi deux variables fortement corrélées sont très proches l'une de l'autre ou au contraire très éloignées selon que la corrélation est directe ou non; deux variables indépendantes sont à distance moyenne, on a d'ailleurs:

$$c(j,j')=\cos(j,j')$$

On peut retenir de l'ACP les idées suivantes:

- Détermination de l'espace qui ajuste au mieux le nuage de points,
- Projection des individus et des variables sur cet espace,
- Interprétation des individus en termes de proximités,
- Interprétation des variables en termes de corrélation.

1.4 Analyse Factorielle des Correspondances (AFC)

L'AFC a un domaine d'application différent. On l'utilise pour des tableaux de contingence où lignes et colonnes jouent des rôles semblables. Lorsqu'un praticien choisit d'effectuer une AFC sur un tableau de données, ce ne sont pas les effectifs bruts des classes qui l'intéressent mais les proportions d'effectifs: autrement dit ce sont les probabilités conditionnelles pour l'individu i d'appartenir à la classe j.

Les p composantes du ième vecteur de Rp vaudront:

$$t(i,j)/t(i,.) \quad (3) \qquad j=1,\ldots,p$$

de même les n composantes du jième vecteur de Rn vaudront:

$$t(i,j)/t(.,j) \quad (4) \qquad i=1,\ldots,n$$

Dans le cas d'une ACP une seule transformation du tableau de données conduisait à une interprétation géométrique différente dans chaque espace (dans l'espace des variables analyse au centre de gravité, dans l'espace des individus projection parallèle à la première bissectrice) et dans le cas d'une AFC deux transformations distinctes (la (3) et la (4)) mènent à une unique interprétation géométrique (analyse au centre de gravité).

La distance entre deux individus i et i' de Rn s'écrit:

$$d(i,i') = \sum_j (p(i,j)/p(i,.) - p(i',j)/p(i',.))^{**2} /p(.,j) \quad (5)$$

On peut démontrer que cette distance vérifie la propriété d'équivalence distributionnelle, c'est à dire deux points voisins dans un espace pourront être considérés comme un seul point correspondant à l'effectif des deux classes réunies. Cette propriété garantit l'invariance des résultats vis à vis de la méthode choisie pour la construction des classes. Par similitude on définit la distance de deux variables j et j' dans Rp:

$$d(j,j') = \sum_i (p(i,j)/p(.,j) - p(i,j')/p(.,j'))^{**}2 \; /p(i,.) \quad (6)$$

Les distances (5) et (6) sont appelées distance du Khi-2.

La part d'informations perdue au cours de la réduction du tableau de données est la même dans les deux espaces. Les vecteurs propres se déduisent les uns des autres par les relations suivantes:

$$ui(m) = \sum p(i,j)/p(i,.)*vj(m)/(lambdam^{**}(1/2)) = \sum_j qj*vj(m)/(lambdam^{**}(1/2)) \quad i \text{ variant de 1 à n} \quad (7)$$

$$vj(m) = \sum p(i,j)/p(.,j)*ui(m)/(lambdam^{**}(1/2)) = \sum_i ri*ui(m)/(lambdam^{**}(1/2)) \quad j \text{ variant de 1 à p} \quad (8)$$

Les relations (7) et (8) peuvent s'exprimer ainsi:

- au coefficient 1/(lambdam**(1/2)) près, les points représentatifs d'un nuage sont sur le mième axe factoriel les barycentres des points représentatifs de l'autre nuage.

Les relations (7) et (8) permettent une interprétation simultanée des deux ensembles sur tout le plan factoriel dans le cas optimum où lambda1 diffère peu de lambda2 et seulement le long des axes factoriels sinon (si un point est proche d'un axe factoriel dans un espace, dans l'autre espace ses coordonnées changent mais les relations (7) et (8) montrent qu'il restera proche du même axe factoriel), ce qui justifie la deuxième appellation de l'AFC comme meilleure méthode de représentation simultanée de deux ensembles de données.

La méthode de l'Analyse Factorielle des Correspondances est:

- Détermination de l'espace qui ajuste au mieux le nuage de points,
- Projection des individus et des variables sur cet espace,
- Interprétation en termes de proximité pour des points d'un même ensemble,
- Interprétation en termes de proximité le long des axes factoriels, et eventuellement sur tout l'espace factoriel, de deux points appartenant à deux nuages distincts.

Les deux méthodes exposées au dessus (ACP,AFC) résument et décrivent l'information présente dans un tableau de données. Ces deux techniques appartiennent aux méthodes descriptives de tableaux. D'autres méthodes cherchent à expliquer les relations observées, c'est à dire à lier une ou plusieurs variables à un ensemble de variables. On appelle ces techniques les méthodes explicatives. Le modèle linéaire et l'analyse discriminante sont deux méthodes explicatives que nous allons présenter.

2 TECHNIQUES EXPLICATIVES

Les techniques descriptives, même parmi les méthodes les plus élaborées, ne permettent pas en général d'analyser complètement un tableau de données, de vérifier un modèle, de tester des hypothèses.... Pour

pallier ces insuffisances il est fréquent d'associer aux techniques descriptives les techniques explicatives dont la régréssion linéaire multiple et l'analyse discriminante sont deux méthodes particulières.

2.1 Regréssion multiple

2.1.1 modèle général

On essaie d'expliquer une variable Y par plusieurs variables x1,...,xp. Le modèle s'écrit:

$$Y = a0+a1*x1+...+ap*xp+Eps \quad (Mp)$$
$$Y = Yp + Eps$$

Il s'agit d'estimer a0,a1,...,ap de telle sorte que l'on minimise la somme des carrés des écarts entre les valeurs observées et les valeurs calculées sur le modèle. Si les variables explicatives x1,...,xp ne sont plus quantitatives mais nominales, le modèle (Mp) est appelé modèle d'analyse de la variance. Eps est un vecteur aléatoire résiduel ayant une matrice de covariance du type sigma2*Id (autrement dit, deux résidus Ei et Ej ont une covariance nulle et même variance sigma2). C'est la seule hypothèse que nous ferons. Pour une réalisation i le modèle (Mp) s'écrit:

$$Yi = a0+a1*x1(i)+...+ap*xp(i)+Ei$$

Si nous supposons n réalisations, le modèle (Mp) s'écrit sous forme matricielle:

$$\begin{bmatrix} Y1 \\ Y2 \\ \\ \\ Yn \end{bmatrix} = \begin{bmatrix} 1 & x1(1) & x2(1) & & xp(1) \\ 1 & x1(2) & x2(2) & & xp(2) \\ & & & & \\ & & & & \\ 1 & x1(n) & x2(n) & & xp(n) \end{bmatrix} \begin{bmatrix} a0 \\ a1 \\ \\ \\ ap \end{bmatrix} + \begin{bmatrix} E1 \\ E2 \\ \\ \\ En \end{bmatrix}$$

soit sous forme condensée

$$Y = Xa + E$$

où E est un vecteur aléatoire d'espérance nulle et de matrice de variance-covariance égale à sigma2 fois la matrice identité de Rn. On démontre que l'estimateur de a qui minimise la somme des carrés des écarts (Σ(Ei**2)) vaut :

$$\hat{a} = (X'X)^{**}(-1)X'Y$$

Or â est lui même un vecteur aléatoire d'espérance a et de variance sigma2*(X'X)**(-1). Dans cette dernière égalité on estime sigma2 inconnue par s2:

$$s2 = 1/(n-p-1) * (Y-X\hat{a})'(Y-X\hat{a})$$

A noter que â et s2 sont des estimateurs sans biais de a et de sigma2.

2.1.2 modèle gaussien

On est souvent conduit à supposer le modèle gaussien. Cette hypothèse supplémentaire permet de construire des intervalles de confiance des coefficients et de batir des tests d'hypothèses. L'hypothèse supplémentaire s'écrit: Ei suit une loi normale de paramètres N(0,sigma2); soit sous forme matricielle E suit une loi normale Nn(0,sigma2*Idn) où Nn signifie un vecteur gaussien à n dimensions et Idn est la matrice identité de Rn. Alors les résultats précédents deviennent:

```
â ∿ Nn(a,sigma2*(X'X)**(-1))
(n-p-1) * s2/sigma2 ∿ Khi-2(n-p-1)
â et sigma2 sont indépendants
```

1: intervalle de confiance

```
Taj =  <(âj-aj)/sigmaaj> / <saj/sigmaaj>
    = N(0,1) /  <(saj**2)/(sigmaaj**2)>**(1/2)
    = N(0,1) /  <Khi-2(n-p-1)/(n-p-1)>**(1/2)
    = t(n-p-1)
```

Ainsi Taj suit une loi de Student à (n-p-1) degrés de liberté. On peut construire un intervalle de confiance au niveau alpha de la façon suivante:

```
P( -t(n-p-1,alpha/2) < Taj < t(n-p-1,alpha/2) ) = 1-alpha
```

expression où alpha est défini par l'utilisateur et t(n-p-1,alpha/2) se lit dans une table de Student. Une fois t(n-p-1,alpha/2) déterminé, l'intervalle de confiance du coefficient aj est Iaj:

```
Iaj = <âj-saj*t(n-p-1,alpha/2) , âj+saj*t(n-p-1,alpha/2)>
```

2: tests d'hypothèses

Lors de l'élaboration d'un modèle de régression linéaire multiple, on est souvent conduit à se poser les trois questions suivantes:

a) existe t-il une association entre la variable Y et l'ensemble des variables x1,...xp ?
b) la variable xj a-t-elle une influence sur le modèle (Mp) ?
c) un groupe de variables a-t-il une influence sur le modèle (Mp) ?

a) on montre que le coefficient de corrélation multiple, défini comme le rapport de la variance expliquée par le modèle de régression sur la variance totale, est égal au carré du coefficient de corrélation des variables Y et a0+a1*x1+...+ap*xp, c'est à dire au carré du cosinus de l'angle formé par ces deux vecteurs. Plus ce coefficient de corrélation multiple tend vers 1, plus le modèle de régression est satisfaisant.

$$R^{**}2 = \text{variance expliquée / variance totale}$$
$$= \Sigma(\hat{y}i - Y.)^{**}2 / \Sigma_i (Yi - Y.)^{**}2$$
$$= \text{corr}(Y,\hat{y})^{**}2$$
$$= \cos(Y,\hat{y})^{**}2$$

expressions où Yi est une réalisation du vecteur Y, $\hat{y}i$ est l'estimation de Yi au moyen du modèle de régression et Y. est la moyenne de toutes les réalisations.

b) la variable xj n'aura pas d'influence sur le modèle (Mp) si nous montrons que son coefficient aj est nul ce qui revient à tester l'hypothèse HO (aj=0) contre l'hypothèse H1 (aj≠0). On déduit du paragraphe précédent la règle de décision qui est d'accepter l'hypothèse HO au niveau alpha si:

$$0 \in \langle \hat{a}j - s\hat{a}j * t(n-p-1,\text{alpha}/2) \ , \ \hat{a}j + s\hat{a}j * t(n-p-1,\text{alpha}/2) \rangle$$

c) Un groupe de (p-r) variables n'aura pas d'influence sur le modèle (Mp) si (p-r) coefficients sont simultanément nuls, ce qui revient à tester l'hypothèse HO (a(r+1)=...=ap=0) contre l'hypothèse H1 (l'un au moins des aj, pour j variant de (r+1) à p, est non nul). Le modèle (Mr) s'écrit:

$$Y = a0 + a1 * x1 + \ldots + ar * xr + E \quad (Mr)$$
$$Y = Yr + E$$

Il sera satisfaisant si une distance, notée d, est suffisamment petite. Notons SCr(p) le carré de la norme du vecteur résiduel issu du modèle (Mp), de même SCr(r) pour le modèle (Mr). Posons:

$$d = SCr(r) - SCr(p)$$
$$F = (d/(p-r))/(SCr(p)/(n-p-1))$$

On montre que si l'hypothèse HO est vraie alors F suit une loi de Fischer à (n-p-1,p-r) degrés de liberté. A alpha fixé, on lit sur une table de Fischer la valeur de F(n-p-1,p-r,alpha) en décidant d'accepter le modèle (Mr) si F<F(n-p-1,p-r,alpha) et de le rejeter sinon.

2.2 Analyse discriminante

Nous finissons la présentation des différentes techniques statistiques employées dans l'étude par deux méthodes de discrimination. Dans les deux cas les données se présentent sous la forme d'un tableau à n lignes (les individus) et (p+1) colonnes (les variables). Parmi les variables, on distingue une variable Y à expliquer, ce sera une variable qualitative à k modalités permettant de répartir les individus en k classes (W1,...,Wk) où k groupes (Gr1,...,Grk), et p variables quantitatives. Par exemple la variable Y peut être: origine des erreurs d'une procédure. Elle possèdera cinq modalités: cahier des charges, spécifications fonctionnelles, spécifications organiques, codage, sans erreur. Les p dernières variables xj sont des variables quantitatives qui représentent les p mesures effectuées sur chaque procédure.

L'analyse discriminante présente deux étapes. L'étape descriptive au cours de laquelle on vérifie que les données relevées sur les individus permettent une bonne séparabilité des différents groupes. L'étape décisionnelle constitue l'essentiel de l'analyse discriminante et permet,

les groupes étant bien distincts, de déterminer le groupe ou la classe la plus probable d'affectation d'un nouvel individu.

Le problème de la discrimination est toujours de construire dans l'espace des variables k régions R1,...Rk qui doivent vérifier au mieux:

- beaucoup d'individus de Wi ont leur vecteur de mesures dans Ri,
- peu d'individus hors de Wi ont leur vecteur de mesures dans Ri.

2.2.1 Méthode de discrimination paramétrique

La méthode de discrimination linéaire paramétrique est une technique de discrimination très usuelle car elle est conceptuellement facile à mettre en oeuvre. Au moyen du fichier de données initial on applique la méthode de l'échantillon test en réalisant un fichier apprentissage et un fichier test. A l'aide du premier fichier on élabore une règle d'affectation des individus que l'on valide sur le fichier test.

a) position du problème

L'analyse linéaire discriminante détermine un repère dans l'espace des variables tel qu'en projection sur les nouveaux axes les points moyens des groupes se séparent le plus possible avec en même temps un regroupement des individus de chaque groupe autour de son point moyen. On cherche un axe D de vecteur directeur u tel que la variance intergroupe u'Bu soit maximale et la variance intragroupe u'Wu soit minimale (la dispersion totale est la somme de la dispersion intergroupe et intragroupe). On cherche à maximiser l'expression Q(u):

$$Q(u) = u'Bu/u'Wu$$

La dérivée en u de cette expression implique:

$$Bu*u'Wu = Wu*u'Bu$$

Supposons W inversible et notons:

$$p = u'Bu/u'Wu$$

Le problème de l'analyse discriminante revient à chercher le vecteur u solution de l'equation:

$$W^{**}(-1)*B*u = pu \quad \text{pour p maximal}$$

Nous pouvons formuler le problème de l'analyse discriminante autrement (<LeMoTa>;p13) :

> faire une analyse linéaire discriminante revient à effectuer une ACP à partir des points moyens des k groupes repérés dans l'espace Rp par la métrique W**(-1).

La distance d'un individu x0 à un groupe G1 est alors:

$$<d(xO,Gl)>**2 = (xO-gl)'*(Wl)**(-1)*(xO-gl)$$

où gl et Wl représentent le point moyen et la matrice de variance covariance du lième groupe. Un nouvel individu xO sera affecté au groupe Gl si et seulement si:

$$d(xO,Gl) = \inf_i d(xO,Gi) \quad i \text{ variant de 1 à k}$$

2.2.2 Méthode de discrimination non paramétrique

Reprenons les notations précédentes. L'idée directrice de la discrimination non paramétrique n'est plus de séparer le plus possible les classes les unes des autres et de regrouper le mieux possible les individus d'une même classe autour de leur point moyen. L'idée fondamentale est de minimiser le coût de mauvaise classification. La règle d'affectation d'un individu à un groupe se fait en fonction de la position de cet individu par rapport à une frontière qui ne dépend maintenant que d'une variable.

Notons c(Wi,Rj) le coût de ranger un individu dans la jième classe alors qu'il appartient à la ième. Le coût moyen de mauvaise classification (risque de Bayes) vaut:

$$\sum_{ij} c(Wi,Rj)*p(Wi,Ri)$$

Remarquons que c(Wi,Ri)=0. Les c(Wi,Rj) sont des paramètres spécifiés par l'utilisateur. Les méthodes faisant décroitre ce risque sont des méthodes bayesiennes. Présentons les principes de la méthode dans le cas de deux familles et d'une seule variable.

Soit W la population totale, W1 et W2 les deux familles avec n=CardW1 et N-n=CardW2. Dans le cas où une seule variable décrit la population, on pose f1(x) et f2(x) les densités continues de probabilité des deux familles et F1(x) et F2(x) les fonctions de répartition correspondantes. On note P1 (respectivement P2) la probabilité à priori d'appartenir à la famille W1 (respectivement W2). On a bien sûr:

$$P1+P2 = 1$$

Soit Z(n+1) un individu supplémentaire, on veut classer Z(n+1) dans W1 ou W2 à partir de l'echantillon Z1,...,Zn en utilisant le point de coupure c de la manière suivante:

si Z(n+1)<c il est affecté à la population inférieure, par exemple W1,
si Z(n+1)>c il est affecté à la population supérieure, soit W2.

On choisit c de façon à minimiser le risque de Bayes de mauvaise classification. On fera l'hypothèse suivante:

$$P1*c(1,2) = P2*c(2,1)$$

En utilisant un point de coupure quelconque Z pour séparer les deux classes, on a le risque de Bayes correspondant:

$$R(Z) = c(1,2)*p(W1,R2)+c(2,1)*p(W2,R1)$$
$$R(Z) = c(1,2)*p(W1)*p(R2|W1)+c(2,1)*p(W2)*p(R1|W2)$$
$$R(Z) = c(1,2)*P1*\int f1(x)dx+c(2,1)*P2*\int f2(x)dx$$
$$R(Z) = c(1,2)*P1-c(1,2)*P1*F1(Z)+c(2,1)*P2*F2(Z)$$
$$R(Z) = c(1,2)*P1*(1-F1(Z)+F2(Z))$$

Minimiser R(Z) revient à maximiser F1(Z)-F2(Z). Comme on ne connait pas à priori la population inférieure et la population supérieure, le point qui minimise le risque de Bayes est le point c donné par la relation suivante:

$$D(c) = \sup_z D(Z) = \sup | F1(Z)-F2(Z) |$$

La quantité D(c) n'est autre que la distance de Kolmogorov-Smirnov appliquée aux deux distributions, c'est une mesure bien connue de la séparabilité de deux fonctions de répartition.
En pratique F1 et F2 ne sont pas connues. On les estime par les fonctions de répartition empiriques.

$$\begin{cases} \hat{F}1(x) = 0 & x<x(1,1) \\ \hat{F}1(x) = k/n & x(1,k)<x<x(1,k+1) \\ \hat{F}1(x) = 1 & x(1,n)<x \end{cases}$$

de même

$$\begin{cases} \hat{F}2(x) = 0 & x<x(2,1) \\ \hat{F}2(x) = k/(N-n) & x(2,k)<x<x(2,k+1) \\ \hat{F}2(x) = 1 & x(2,N-n)<x \end{cases}$$

expressions où x(i,k) est le kième point de la famille Wi, les points étant rangés par ordre croissant. L'hypothèse P1*c(1,2)=P2*c(2,1) permet de montrer que le rique estimé $\hat{R}(\hat{c})$ converge en probabilité vers le risque de Bayes R(c).

L'extension de la méthode précédente au cas multidimensionnel est d'effectuer la coupure pour la variable où la distance de Kolmogorov-Smirnov entre les deux distributions est la plus grande. L'algorithme peut être résumé ainsi:

$$D(cj) = \sup_z | F1j(Z) - F2j(Z) |$$

expression où F1j et F2j représentent les fonctions de répartition de la variable j sur les familles W1 et W2. On effectue la coupure pour la variable j* telle que:

$$D(cj^*) = \sup_j D(cj)$$

La coupure se fait au point cj*. Un test d'arrêt termine la procédure de discrimination. Il est basé sur une sous-représentation d'une classe ou des deux classes simultanément. Un critère de qualité de la discrimination effectuée à l'étape e est donné; il traduit la part de mauvaise classification.
La discrimination non paramétrique dans le cas multi-classes n'est pas présentée ici car elle dépasse le cadre de cette étude.

Vocabulaire: la discrimination exposée au paragraphe 2.2.1 est paramètrique car les classes obtenues à postériori peuvent être instables après application d'une fonction bijective sur les variables. Par contre la discrimination bayesienne est non paramètrique car la classification, au moyen d'une seule variable, est indépendante de toute modification de cette variable par une fonction bijective.

BIBLIOGRAPHIE

BIBLIOGRAPHIE

<AlFe82> J.L Albin , R.Ferréol
Collecte et analyse de mesures de logiciel
AFCET / Premier Colloque de Génie Logiciel
Paris
8-10 Juin 1982

<AlPo82> J.L Albin , D.Potier
Analyse des données d'erreurs et des mesures de complexité
Projet Pilote SURF : Bilans et Perspectives
Paris , Agence de l'Informatique
20-22 Janvier 1982

<BaZw79> A.Baker , S.Zweben
A comparison of measures of control flow complexity
IEEE pp 695-701
1979
/ complexity measures / control flow /

<BiPo80> A.Bilodeau, M.Poize
Outil d'aide à la conception de tests et de documentation de programme LTR
Agence de l'Informatique, Projet Pilote SURF
Fevrier 1980

<Boh75> R.Bohrer
Halstead's criterion ans statistical algorithms
Proc. Eighth Annual Computer Science/Statistics Interface Symp.
UCLA, Los-Angeles
Fevrier 1975

<CeLe80> G.Celeux , Y.Lechevallier
Méthodes de discrimination non paramétrique asymptotiquement efficaces au sens de Bayes
Rapport de recherche no 52 INRIA 78150 Le Chesnay
Decembre 1980

<Ch79> N.Chapin
A measure of software complexity
National Computer Conference
(AFIPS), pp 995-1002
1979

<Cu79> K.Culik
The cyclomatic number and the normal number of programs
Sigplans Notices, Vol 14, pp 12-17
Avril 1979

<CuSh79> B.Curtis , S.Sheppard , P.Milliman
Third time charm : stronger prediction of programmer performance by software complexity metrics .
Proc. of the Fourth International Conference on Software Engineering, pp 356-360
IEEE Munich (RFA)
Septembre 1979
/ programmer / complexity /

<DACS79> Data and Analysis Center for Software
RADC/ISISI (Mrs Lorraine Duvall)
Griffiss Air Force Base, NY 13441
Quantitative software models
Mars 1979

<DGHu79> V.Donzeau-Gouge , G.Huet , G.Kahn , B.Lang
Introduction au système Mentor et à ses applications
Comptes rendus des journées Francophones sur la certification du logiciel, pp114-143
Genève, SUISSE
Janvier 1979

<DuIn79> A.M.Dussaix , J.P.Indjehagopian
Méthodes statistiques appliquées à la gestion
Les éditions d'organisation rue Rousselet 75007 Paris
Collection ESSEC 1979

<ECA80> ECA-Automation
Outil d'aide à la conception de tests et de documentation de programmes LTR
Projet SURF
Agence de l'informatique, La Défense 92
1980

<Els76> J.L.Elshoff
Measuring commercial PL/1 programs using Halstead' criteria
Sigplans notices Vol 7 , no 5, pp 38-46
Mai 1976
/ lenght / estimated lenght /

<Els76> J.L.Elshoff
An analysis of some commercial PL/1 programs
IEEE Trans. of Software Engineerings, Vol SE-2, n2, pp 113-120
Juin 1976
/ experiments / PL/1 /

<FeFo79> A.Feuer , E.Fowlkes
Relating computer program maintainability to software measures
IEEE Proc. of the 1979 National Computer Conference, pp 1003-1012
Juin 1979
/ experiment / error / complexity / cluster analysis /

<FiLo79> A.Fitzsimmons , L.Love
A review and evaluation of the Software Science
ACM computing surveys, Vol 10, n1, pp 3-18
Mars 1978
/ complexity / reliability /

<Fit80> G.Fitsos
Vocabulary effects in Software Science
IBM technical report TR03.082
IBM Corp., San Jose, CA
Janvier 1980
/ operators / operands /

<FuHa76> Y.Funami , M.Halstead
A software physics analysis of Akiyama's debugging data
Proc. on the Symp. on Computer Software Engineering, pp 133-138
Polytechnic Institute of New-York, NY
Mai 1976

<Ha77> M.Halstead
Elements of Software Science
Elsevier North-Holland Company
1977
/ software science /

<Han78> W.Hansen
Measurement of program complexity by the pair
Sigplans notices, Vol 13, n3, pp 29-33
Mars 1978
/ cyclomatic / complexity /

<Hu73> E.Hunt , al
Individual differences in cognition : a new approach to intelligence
Advances in learning and motivation G.Bower(Ed.), Vol 7
Academic Press, NY
1973

<Kn71> D.Knuth
An empirical study of FORTRAN programs
Software practice and experience, Vol1, pp 105-133
Novembre 1971
/ FORTRAN / optimization / efficiency / compiler /

<LeFe75> L.Lebart , J.P.Fenelon
Statistique et informatique appliquées
Dunod

<LeMoTa> L.Lebart , A.Morineau , N.Tabard
Techniques de la description statistique
Dunod

<McC76> T.Mac Cabe
A complexity measure
IEEE Trans. of Software Engineering, Vol SE-2, n4, pp 308-320
Novembre 1976
/ cyclomatic / complexity / control flow /

<MeRa79> G.Memmi , J.Cl.Rault
Etude sur la fiabilite des logiciels temps reel des missions spatiales
European space agency (contrat: n 3326/77/D/JS/SC/)
Janvier 1979
/ tests / materiel / logiciel /

<MoBr77> R.W.Motley , W.D.Brooks
Statistical prediction of programming errors
RADC-TR-77-175
IBM Corp, Federal Systems Division,
1601 North Kent Street, Arlington, VA 22209
1977

<My77> G.Myers
An extension to the cyclomatic measure of program complexity
Sigplans notices, Vol 12, n 10, pp 61-64
Octobre 1977
/ cyclomatic / complexity /

<Pa80> M.Paige
Wang Laboratories Inc, Lowell, Massachusetts 01851
A metric for software test planning
Proc. COMPSAC pp 499-504
1980

<Pi76> S.Pimont
Une evaluation de la fiabilité du logiciel s'appuyant sur une analyse de la structure et du comportement des programmes
These de troisieme cycle
Institut de programmation, Paris
Decembre 1976
/ tests / materiel /

<PoA182> D.Potier , J.L Albin , R.Ferrèol , A.Bilodeau
Experimenting with computer software complexity and reliability
Intern. Symposiu on Fault-Tolerant Computing FTCS-12
IEEE
Santa Monica , Californie
21-24 Juin 1982

<PoA182> D.Potier , J.L. Albin , R.Ferrèol , A.Bilodeau
Experimenting with computer software complexity and reliability
6Th Intern. Conference on software engineering
IPS, ACM, IEEE, NBS
Tokyo
13-16 Septembre 1982

<Ra80> J.Cl.Rault
Le controle de la qualité et l'evaluation de la fiabilité des produits logiciels
Journées d'études: Evaluation de la qualité du logiciel
INRIA 78150 Le Chesnay
28-29 Avril 1980
/ tests / materiel / logiciel / bibliographie /

<ScHo79> N.Schneidewind , H.Hoffmann
An experiment in software error data collection and analysis
IEEE Trans. on Software Engineering, Vol SE-5, n3, pp 276-286
Mai 1979
/ error / complexity / experiment /

<Sch78> A.Schroeder
Multidimensional data analysis as a tool for the study of computer systems
INRIA 78150 Le Chesnay
1978

<Sm79> C.Smith
Practical applications of Software Science
IBM technical report TR03.067
IBM Corp., San Jose, CA
Juin 1979
/ experiment / software science /

<Sm80> C.Smith
A software science analysis of IBM products
IBM technical report TR03.081
IBM Corp., San Jose, CA
Janvier 1980
/ experiment / complexity /

<Sm81> C.Smith
The application of Halstead software science difficulty measure to a set of programming projects
IBM technical report TR03.125
IBM Corp., San Jose, CA
Janvier 1981
/ experiment / complexity / reliability / difficulty /

<St66> J.Stroud
The fine structure of psychological time
Annals of New-York academy of sciences
1966

<Ste66> S.Sternberg
High-speed scanning in human memory
Science 153
1966

<Su81> T.Sunohara , al
Program complexity measure for software developpment management
Proc. of the Fifth International Conference on Software Engineering, pp 100-106
IEEE, San Diego, CA
9-12 Mars 1981
/ complexity measures / cyclomatic / effort /

<Th78> T.Thayer , M.Lipow , E.Nelson
Software reliability
TRW series of software technology, Vol 2
TRW Systems & Energy
Elsevier North-Holland Company
1978

<Wo79> M.R. Woodward, M.A. Hennell, D. Hedley
A measure of control flow complexity in program text
IEEE Trans. on Software Engineering, Vol SE-5, n 1, pp 45-50
Janvier 1979

Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique

ISBN 2 - 7261 - 0315 - 4